Découverte de services et collaboration au sein d'une flotte hétérogène et hautement dynamique d'objets mobiles communicants autonomes

Vincent Autefage

université de BORDEAUX

LaBRI

THÈSE

présentée à

L'UNIVERSITÉ DE BORDEAUX

École Doctorale de Mathématiques et Informatique

par

Vincent AUTEFAGE

pour obtenir le grade de

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Découverte de services et collaboration au sein d'une flotte hétérogène et hautement dynamique d'objets mobiles communicants autonomes

---

Soutenue le 26/10/2015

Membres du jury

Frédéric Guinand	Professeur	Université du Havre	Rapporteur, Président
Pascal Bouvry	Professeur	Université du Luxembourg	Rapporteur
Arnaud Casteigts	Maître de conférences	Université de Bordeaux	Examinateur
Véronique Serfaty	Docteur	DGA	Examinateur
Serge Chaumette	Professeur	Université de Bordeaux	Directeur de thèse
Damien Magoni	Professeur	Université de Bordeaux	Directeur de thèse
Jean-Marc Grolleau	Industriel	Cluster AETOS - Thales	Invité
Tristan Mehamli	Industriel	Mugen	Invité

Remerciements

Le doctorat est un chemin semé d'embûches. Ma conviction est que les nombreuses épreuves mais aussi les rencontres qui se présentent sur ce chemin sinueux permettent, du moins je l'espère, de nous rendre meilleurs.

Je souhaite tout d'abord remercier Frédéric Guinand et Pascal Bouvry pour avoir accepté de rapporter cette thèse. Je remercie également l'ensemble des membres du jury pour avoir accepté de m'écouter pendant 45 minutes et pour les discussions que nous avons eues par la suite.

Merci à mes deux directeurs, Damien et Serge, pour leur accueil, leur confiance, leur soutien ainsi que pour leurs conseils. Je tiens également à remercier la DGA ainsi que la Région Aquitaine pour avoir accepté de financer ces travaux.

Merci à Tom B. pour avoir relu scrupuleusement ce manuscrit et pour ses conseils avisés. Merci à Martin P. pour nos discussions fructueuses. Je remercie également l'ensemble des membres du LaBRI et de l'université de Bordeaux (permanents, non permanents, chercheurs, ingénieurs, personnels administratifs et techniques) avec qui j'ai pu avoir des échanges constructifs tout au long de ces années. Merci notamment aux locataires du bureau 124 qui ont su créer une ambiance agréable et propice aux discussions. Une pensée également pour l'ensemble des collègues avec qui j'ai effectué mes enseignements (IUT, UF Informatique, UF Mathématiques et Interactions).

D'un point de vue plus personnel, je remercie sincèrement Noémie pour m'avoir soutenu et encouragé durant toutes ces années. Je remercie enfin ma famille et mes amis pour leur patience.

```
groups = [reviewers, jury, advisors, financers, colleagues, wife, family, friends]
for people in groups :
    me.thanks(people)
```

Information

Cette thèse a été co-financée par la **Direction Générale de l'Armement** [1] ainsi que par la **Région Aquitaine** [2].

Elle a été réalisée sous la direction de **Serge Chaumette** et **Damien Magoni** dans le cadre des groupes de recherche **MUSe** [3] et **COMET** [4] du **LaBRI** [5].

1. http://www.defense.gouv.fr/dga
2. http://aquitaine.fr
3. http://muse.labri.fr
4. http://comet.labri.fr
5. http://www.labri.fr

Découverte de services et collaboration au sein d'une flotte hétérogène et hautement dynamique d'objets mobiles communicants autonomes

Résumé

Les systèmes autonomes sont des objets mobiles communicants capables de réaliser un certain nombre de tâches sans intervention humaine. Le coût (*e.g.* argent, poids, énergie) de la charge utile requise pour effectuer certaines missions est parfois trop important pour permettre aux engins d'embarquer la totalité des capacités nécessaires (*i.e.* capteurs et actionneurs). Répartir ces capacités sur plusieurs entités est une solution naturelle à ce problème. Un tel groupe d'entités constitue une flotte à laquelle il devient nécessaire de fournir un mécanisme de découverte permettant aux différents engins de partager leurs capacités respectives afin de résoudre une mission globale de façon collaborative. Ce mécanisme, outre l'affectation des tâches, doit gérer les conflits et les pannes potentielles qui peuvent survenir à tout moment sur tout engin de la flotte. Fort de ces constations, nous proposons un nouveau mécanisme collaboratif nommé AMiRALE qui apporte une solution aux problèmes ci-dessus pour les flottes hétérogènes d'engins mobiles autonomes. Notre système est entièrement distribué et repose uniquement sur des communications asynchrones. Nous proposons également un nouvel outil nommé NEmu permettant de créer des réseaux virtuels mobiles avec un contrôle important sur les propriétés de la topologie du réseau ainsi que sur la configuration des nœuds et des inter-connexions. Cet outil permet la réalisation d'expérimentations réalistes sur des prototypes d'applications réseaux. Enfin, nous proposons une évaluation de notre système collaboratif AMiRALE au travers d'un scénario de nettoyage de parc utilisant une flotte autonome de drones et de robots terrestres spécialisés.

Mots clés : Découverte de services, collaboration, systèmes autonomes, flottes dynamiques.

Service Discovery & Collaboration in a Heterogeneous & Highly Dynamic Swarm of Mobile Communicating & Autonomous Objects

Abstract

We call autonomous systems, mobile and communicating objects which are able to perform several tasks without any human intervention. The overall cost (including price, weight and energy) of the payload required by some missions is sometimes too important to enable the entities to embed all the required capabilities (*i.e.* sensors and actuators). This is the reason why it is more suitable to spread all the capabilities among several entities. The team formed by those entities is called a *swarm*. It then becomes necessary to provide a discovery mechanism built into the swarm in order to enable its members to share their capabilities and to collaborate for achieving a global mission. This mechanism should perform task allocation as well as management of conflicts and failures which can occur at any moment on any entity of the swarm. In this thesis, we present a novel collaborative system which is called AMiRALE for heterogeneous swarms of autonomous mobile robots. Our system is fully distributed and relies only on asynchronous communications. We also present a novel tool called NEmu which enables to create virtual mobile networks with a complete control over the network topology, links and nodes properties. This tool is designed for performing realistic experimentation on prototypes of network applications. Finally, we present experimental results on our collaborative system AMiRALE obtained through a park cleaning scenario which relies on an autonomous swarm of drones and specialized ground robots.

Keywords : Service Discovery ; Collaboration ; Autonomous Systems ; Dynamic Swarms.

Laboratoire Bordelais de Recherche en Informatique (LaBRI - UMR 5800)
Université de Bordeaux
Bâtiment A30, 351 cours de la Libération, 33405 Talence Cedex, France

Table des matières

Liste des figures

Introduction

Un *objet communicant*, plus communément appelé *objet connecté*, est un dispositif comprenant des capteurs, des actionneurs, des processeurs ainsi qu'un ou plusieurs mécanismes de communication et ayant pour objectif de fournir un certain nombre de services [1]. Ces services peuvent aussi bien être de nature matérielle (*e.g.* une fonction aspirateur) que logicielle (*e.g.* une collecte de données). Le nombre de ces objets communicants n'a cessé de croître depuis quelques années, passant de 200 millions en 2000 à 10 milliards en 2013 et devrait, selon les estimations les plus récentes, atteindre les 30 à 80 milliards d'ici 2020 [2, 3, 4]. Parmi ces objets communicants, on distingue une famille particulière : celle des robots. Ces engins de tout type (*i.e.* terrestres, aériens, marins, etc.) ont également connu une forte expansion tant dans le domaine civil que militaire [5] ; l'état Français a identifié la robotique comme une nouvelle révolution industrielle, comparable à celle d'Internet, et se fixe pour objectif de compter parmi les cinq nations leaders dans ce domaine d'ici 2020 [6].

Certains robots sont conçus pour communiquer et interagir au sein d'un groupe. Pour distinguer de tels groupes, on parle de *flotte* ou d'*essaim*[1] [7]. Une flotte est dite autonome dès lors que les engins qui la composent ont la faculté de s'autogérer au sens large (*i.e.* sans intervention extérieure). On parle plus généralement de *système autonome* (*Unmanned System* - UMS) [8]. Par ailleurs, une flotte est considérée comme hétérogène lorsque les robots qui la composent sont de nature différente et/ou disposent de capacités distinctes. Dans une telle flotte, il est nécessaire d'implémenter un mécanisme de découverte permettant à ses membres de partager leurs capacités et ainsi de collaborer pour mener à bien une mission commune. Nous appelons une capacité ainsi partagée un *service*. Cette thèse s'inscrit dans ce contexte des flottes autonomes, hétérogènes et fortement dynamiques (*i.e.* volatiles et mobiles). La difficulté fondamentale est que n'importe quel robot peut rejoindre ou quitter une flotte à tout moment, en raison de sa mobilité ou de pannes éventuelles, ce qui implique des changements récurrents et imprévisibles de la topologie du réseau sous-jacent.

1. La distinction précise entre ces deux termes sera abordée dans la première partie de la thèse.

L'enjeu scientifique majeur de cette thèse est de proposer un mécanisme permettant non seulement la découverte des services dans une flotte conforme aux spécificités décrites ci-dessus, mais également l'exploitation de ces services au travers d'un système collaboratif résistant à des changements topologiques fréquents.

La première partie de cette thèse est un état de l'art traitant des différents aspects du problème. Ainsi, nous détaillons tout d'abord les principales caractéristiques et les architectures possibles de flottes autonomes et hétérogènes. Nous proposons également une analyse des principaux mécanismes de découverte de services existant et précisons leur adéquation ou non aux flottes auxquelles nous nous intéressons. Nous étudions les différents mécanismes de communication au sein d'une flotte et l'impact des modèles de mobilité des engins sur de tels systèmes. Enfin, nous présentons les principaux modèles d'interaction permettant d'appréhender le comportement des engins (individuellement et globalement) qui composent une flotte autonome et hétérogène.

Dans la seconde partie de ce document, nous proposons un système collaboratif original entièrement distribué que nous appelons AMiRALE (*Asynchronous Missions Relay for Autonomous and Lively Entities*). Notre système permet la découverte des services disponibles dans une flotte et l'exploitation des services ainsi découverts. Nous proposons une description complète de notre système ainsi qu'une modélisation basée sur la notion de ré-étiquetage de graphes dynamiques.

Dans la dernière partie du document, nous présentons des résultats expérimentaux reposant sur un scénario de nettoyage de parc à l'aide de robots terrestres spécialisés et de drones aériens. Nous déterminons ainsi les capacités minimales de stockage et de communication nécessaires au fonctionnement de notre système collaboratif. Ces résultats ont été obtenus non seulement par simulation mais aussi par émulation. Pour ce dernier aspect, nous proposons un nouvel outil appelé NEmu (*Network Emulator for mobile universes*) permettant la création de réseaux mobiles hautement configurables composés de machines virtuelles. Enfin, nous présentons la première étape d'un démonstrateur, fruit de la collaboration entre le LaBRI et l'entreprise *Mugen*, illustrant notre scénario de nettoyage de parc.

Enfin, nous concluons ce manuscrit en résumant nos contributions et en abordant les perspectives de ces travaux.

I État de l'art

CHAPITRE 1

Flottes autonomes

Sommaire

1.1 Systèmes autonomes

Un *système autonome* (*Unmanned System* - UMS) est un engin mobile physique inhabité capable d'effectuer un ensemble de tâches sans nécessiter d'intervention humaine [8]. Dans cette section, nous détaillons les types de systèmes autonomes existants ainsi que leur degré d'autonomie respectif.

1.1.1 Types de systèmes autonomes

Les systèmes autonomes peuvent exister sous plusieurs formes. Ils diffèrent principalement par leurs moyens de déplacement et de manœuvre. Les différences peuvent également porter sur leur autonomie (au sens énergétique) ainsi que sur les capacités d'emport de l'engin.

Systèmes autonomes aériens (UAV/UAS/RPAS)

Un UAV (*Unmanned Aerial Vechicle*) ou *drone*, est un UMS volant. Le terme UAS (*Unmanned Aerial System*) englobe à la fois l'engin et son système de contrôle. Le terme RPAS (*Remote Piloted Aircraft System*) désigne les UAS télé-opérés (*i.e.* pilotés à distance par un humain) [9].

Les voilures fixes reposent sur un système d'ailes statiques (généralement au nombre de deux) et sont propulsés par une énergie de type électrique ou thermique. Ces types de voilures sont principalement utilisés pour des vols à haute altitude (*i.e.* plusieurs milliers de pieds) et/ou de longue durée (*i.e.* plusieurs heures). Elles peuvent atteindre des vitesses significatives (*i.e.* plusieurs centaines de kilomètres par heure) mais sont limitées dans leur liberté de mouvements. En effet, les voilures fixes sont incapables d'effectuer des descentes (resp. des ascensions) verticales ou de prendre des virages trop serrés en raison du risque de décrochage de l'appareil. Nous pouvons citer en exemple le *MQ-9* de l'entreprise américaine *General Atomics* qui est un drone de surveillance et de combat à voilure fixe utilisé à la fois par les armées américaine et française [10, 11, 12]. Le *DT-18* de l'entreprise française *Delair Tech* est également un drone à voilure fixe mais dont les applications sont exclusivement tournées vers le domaine civil, notamment l'inspection et l'analyse de terrain [13].

Les voilures tournantes utilisent un ensemble de rotors propulsés généralement par une énergie électrique ou thermique. Ce type de voilures confère aux drones des capacités aérodynamiques beaucoup plus extrêmes que celles des voilures fixes (*e.g.* descentes et montés verticales, virages serrés, pivots). Néanmoins, la portance de l'appareil repose uniquement sur l'action des rotors, ce qui entraîne une consommation énergétique importante limitant son autonomie (au sens énergétique) et le poids maximal de sa charge utile. Par conséquent, les voilures tournantes sont privilégiées pour des vols courts, en basse altitude et où le scénario implique des mouvements complexes mais ne nécessite pas une charge utile trop importante. Nous pouvons citer comme exemple de voilure tournante le *X4* de l'entreprise française *Fly-n-Sense* qui est un quadrirotor principalement dédié aux applications civiles telles que la surveillance des feux de forêts, la sécurité civile ou encore l'agriculture [14]. L'*Ar.Drone2* de l'entreprise française *Parrot* est quant à lui un quadrirotor de loisir [15].

Les aérostats reposent sur la poussée d'Archimède pour assurer leur sustentation. Ces appareils sont capables d'évoluer à très haute altitude (*i.e.* au niveau de la stratosphère) avec une autonomie très importante (*i.e.* de l'ordre de plusieurs jours). Ils sont néanmoins très limités dans leurs capacités de mouvements et sont donc à proscrire pour des scénarios nécessitant une mobilité importante. Le dispositif *Gélule* de l'entreprise française *Phodia* est un exemple d'aérostat utilisé principalement à des fins publicitaires et pour des prises de vues aériennes [16].

Les drones à voilure battante reproduisent le mouvement des ailes de certains insectes ou oiseaux. Ce type de voilure requiert une grande légèreté de l'appareil limitant fortement sa capacité d'emport. Par conséquent, les voilures battantes sont principalement utilisées à des fins de loisir. Dans cette catégorie, il existe par exemple le *BionicOpter* de l'entreprise allemande *Festo* qui est un appareil imitant les mécanismes de vol d'une libellule [17].

Systèmes autonomes terrestres (UGV/UGS)

Un UGV (*Unmanned Ground Vehicle*) est un UMS terrestre. De même que pour les UAS, le terme UGS (*Unmanned Ground System*) englobe à la fois l'engin et son système de contrôle. Ces appareils sont aussi bien utilisés dans le domaine civile que militaire [6, 11].

Un grand nombre d'UGVs utilisent un système de roues pour se déplacer. Le *Vitirover* en est un exemple ; il est principalement utilisé pour l'entretien des plans de vignes [18]. Le *Robco* de la société *R&DTech France* est, quant à lui, utilisé pour l'inspection et la reconnaissance [19]. Les engins de loisir tels que l'*Ollie* de la société *Orbotix* [20] ou le *Jumping Sumo* de *Parrot* [21] font également partie de cette catégorie d'UGVs.

D'autres types d'UGVs reposent sur un mécanisme à chenilles. Un tel système confère à l'engin une plus grande stabilité au détriment des possibilités de manœuvre. Le *Skopy* de *R&DTech France* [22], le *Bulldog* de la société *SDR* [23] ou encore le *SUGV* de l'entreprise américaine *iRobot* [24] sont des exemples de cette catégorie d'UGVs.

Les robots marcheurs représentent également une part importante des UGVs existants. Leur grande manœuvrabilité leur facilite le franchissement des obstacles et leur permet d'évoluer sur des terrains à fort relief. Les robots *BigDog* et *Rise* de la société américaine *Boston Dynamics* illustrent ce type d'UGVs [25, 26]. Le chien *Aibo* de *Sony* [27] et le robot bipède *Nao* de l'entreprise française *Aldebaran* [28] sont quant à eux des exemples de robots marcheurs de loisir.

Les robots rampants mettent en œuvre une technologie communément appelée *soft robotic* [29]. Cette approche se développe depuis plusieurs années principalement pour des applications militaires. Le *Snake* du laboratoire *BRML* [30] s'inspire par exemple de la morphologie et des mécanismes de déplacement du serpent. Le *Meshworm* du *MIT* [31] s'inspire quant à lui des mouvements d'un ver de terre.

Il existe d'autres formes de robots qui utilisent des mécanismes de déplacement différents. Par exemple, le *Sphero* d'*Orbotix* [32] est un robot sphérique roulant sur lui même. Le *Kilobot* [33] de l'*université d'Harvard* [33] utilise de son coté des tiges vibrantes pour se mouvoir.

Systèmes autonomes marins (UMV)

Les UMV (*Unmanned Maritime Vehicle*) regroupent à la fois les véhicules de surface (*Unmanned Surface Vehicle* - USV), appelés aussi ASV (*Autonomous Surface Vehicle*) [34] lorsqu'ils sont autonomes et les engins sous-marins (*Unmanned Undersea Vehicle* - UUV). Ces derniers sont aussi parfois appelés AUV (*Autonomous*

Underwater Vehicle) [35] dans le cas où ils disposent d'une certaine autonomie de fonctionnement. Ces engins, notamment de part leurs contraintes de mouvement et leurs systèmes de communication particuliers, posent de nombreuses problématiques spécifiques qui sortent du cadre de notre étude. Ces engins ne seront donc pas considérés dans ce document.

Systèmes hybrides

Il existe également des engins hybrides, *i.e.* qui disposent des attributs de plusieurs types d'UMS, leur permettant ainsi d'évoluer dans plusieurs milieux. Par exemple, le *Snapdragon Cargo* de l'entreprise américaine *Qualcomm* est à la fois un UAV à voilure tournante et un UGV à chenilles [36]. On peut également considérer le *Rolling Spider* de *Parrot* comme un UMS hybride possédant une voilure tournante et des roues [37]. Tout comme les systèmes marins et sous-marins, les systèmes hybrides ont des propriétés spécifiques, principalement en termes de mobilité, qui sortent du cadre de nos travaux.

1.1.2 Niveaux d'autonomie

Le NIST (*National Institute of Standards and Technology*) introduit quatre modes opératoires pour les systèmes autonomes. Ces modes définissent le niveau d'autonomie des engins.

On considère qu'un UMS est *télé-commandé* lorsque ce dernier nécessite un contrôle continu par un être humain. Dans cette configuration, l'engin doit être commandé à vue dans la mesure où il ne transmet pas les informations nécessaires à un pilotage à distance (*e.g.* télémétrie, retour vidéo). Ainsi, un drone piloté à vue par un opérateur est considéré comme un système télé-commandé. Cette catégorie de drones est compatibles avec les scénarios d'utilisation S1 et S3 définis par la direction générale de l'aviation civile [38, 39, 40].

Un UMS *télé-opéré* nécessite également un contrôle continu de la part d'un être humain. Néanmoins, ce contrôle peut être effectué hors de la vue de l'opérateur, les informations nécessaires étant transmises à l'opérateur par l'intermédiaire des différents capteurs présents sur l'engin télé-opéré. Ces systèmes sont compatibles avec les scénarios d'utilisation S2 et S4 définis par la direction générale de l'aviation civile [38, 39, 40].

Un système est considéré comme *semi-autonome* s'il est capable d'effectuer certaines actions sans intervention humaine (*e.g.* planification de trajectoire). Il n'est néanmoins pas capable de fonctionner de manière indépendante et requiert des commandes externes pour effectuer certaines actions (*e.g.* prise de décisions, allocation de tâches, résolution de conflits). Dans ce contexte, l'opérateur peut être considéré comme un superviseur et non plus comme un pilote. Le drone *eBee* de l'entreprise suisse *senseFly* peut ainsi être considéré comme semi-autonome. Il offre en effet un suivi autonome de plan de vol ainsi que la gestion automatique de prises de vues et de mesures agricoles [41].

Enfin, un système *totalement autonome* n'a besoin d'aucune intervention humaine pour effectuer un ensemble de missions prédéfinies. Un tel système doit être

capable de gérer, dans une certaine mesure, les pannes potentielles qu'il peut rencontrer et d'adapter son comportement en fonction des informations collectées par ses capteurs.

Nous introduisons ici une nouvelle catégorie de systèmes totalement autonomes que nous appelons *système indépendant*. Cette catégorie est inspirée de la vision de l'autonomie développée par Kephart et Chess [42]. Ces derniers définissent l'autonomie comme la capacité d'un système à s'auto-gérer sans nécessiter d'action ou d'intervention extérieure (humaine ou artificielle) pour effectuer un ensemble de missions définies. Or, même si un système totalement autonome n'implique aucune intervention humaine, il peut néanmoins nécessiter un contrôle externe automatisé. Par exemple, la plate-forme de drones utilisée par le laboratoire *GRASP* de l'*université de Pennsylavnie* repose sur une station sol exécutant une chaîne de programmes sous Matlab et ROS [43, 44]. Cette chaîne de traitements permet, entre autres, le calcul des trajectoires et les prises de décisions des engins [45, 46]. Cette observation est également valable pour la plate-forme *Raven* du *MIT* [47] et la *Flying Machine Arena* de l'*ETH de Zurich* [48]. Nous définissons donc ci-dessous un nouveau niveau d'autonomie :

Définition 1.1. *Un système autonome est* ***indépendant*** *si et seulement si l'ensemble des calculs et décisions nécessaires à la réalisation de ses missions sont effectués à bord de l'engin. L'engin ne doit donc pas dépendre d'une entité externe (humaine ou artificielle) pour fonctionner.*

Dans notre étude, nous considérerons uniquement des systèmes indépendants.

1.2 Flottes de systèmes autonomes

Dès lors que plusieurs systèmes autonomes sont utilisés pour réaliser une mission commune, nous appelons *flotte* l'ensemble formé par ces systèmes autonomes [7].

1.2.1 Architectures de contrôle

Dans la littérature, il existe principalement trois architectures pour les flottes d'objets mobiles [49, 50, 51, 52, 53, 54] : centralisée, hiérarchique et décentralisée.

L'architecture *centralisée* repose sur une entité unique qui contrôle (partiellement ou totalement) les engins de la flotte. Cette architecture est à la base des exemples de systèmes autonomes non indépendants présentés dans la section 1.1.2 [45, 46, 47, 48]. L'utilisation d'une entité de contrôle unique fournit une connaissance globale de la flotte permettant ainsi d'optimiser les opérations et les actions de chaque engin. En revanche, la robustesse d'une telle architecture est extrêmement limitée dans la mesure où une rupture de communication entre un des engins de la flotte et la station de contrôle entraînerait un défaut (voire un arrêt) de fonctionnement de celui-ci. Une telle architecture ne peut donc pas être considérée comme tolérante aux pannes [55]. De plus, elle limite fortement la taille maximale de la flotte. En effet, les engins ont besoin de communiquer de façon continue avec l'entité de contrôle afin de pouvoir effectuer leurs missions. Cette entité centrale doit effectuer un certain

nombre d'opérations pouvant être plus ou moins complexes et donc utiliser plus ou moins de ressources et de temps. Une augmentation du nombre d'engins dans la flotte entraîne donc une augmentation de la charge de travail et de communication de cette entité. Par conséquent, celle-ci doit disposer de ressources suffisantes. Ce type de flotte est, à l'heure actuelle, principalement utilisée à des fins artistiques [56, 57, 58]. Iocchi *et al.* discernent deux niveaux de centralisation [51] :

— la *centralisation forte* dans laquelle l'entité de contrôle ne change jamais ;
— la *centralisation faible* dans laquelle l'entité de contrôle peut changer au cours de l'exécution d'un scénario.

Même si la centralisation faible permet une plus grande robustesse face aux éventuelles pannes de l'entité de contrôle (*e.g.* en utilisant un point de contrôle secondaire), elle ne permet pas de supprimer la dépendance entre les engins de la flotte et leur point de contrôle.

L'architecture *hiérarchique* utilise plusieurs entités de niveaux différents pour contrôler les engins de la flotte. Dans ce modèle, une entité de niveau n peut contrôler une partie des entité de niveau $n-1$. Ces entités peuvent éventuellement être elles mêmes des engins appartenant à la flotte. Bien que plus extensible (passage à l'échelle possible) qu'une architecture centralisée, l'architecture hiérarchique est difficilement maintenable en cas de panne sur une entité de niveau élevé. La robustesse aux pannes n'est donc pas assurée par ce type d'architecture.

L'architecture *décentralisée* repose en totalité sur des systèmes autonomes indépendants. Dans ce modèle, chaque entité de la flotte s'auto-gère et n'est contrôlée par aucune entité externe. Cette architecture, contrairement aux deux autres, offre en particulier une forte tolérance aux pannes du fait de l'utilisation exclusive d'engins autonomes indépendants. De plus, elle autorise la flotte à être extensible (*i.e.* ajouter ou retirer des engins de la flotte) car elle ne nécessite pas de communication permanente entre les engins et des entités de contrôle, ni de calculs intensifs sur un ensemble d'entités particulières [49, 50, 54].

En s'inspirant de la définition de flotte auto-organisée donnée par Dudek *et al.* [7], nous définissons une flotte autonome comme suit :

Définition 1.2. *Une flotte est* ***autonome*** *si et seulement si elle ne repose que sur un ensemble de systèmes autonomes indépendants organisés selon une architecture décentralisée.*

Dans cette étude, nous considérerons exclusivement ce type de flotte.

1.2.2 Flottes et essaims

Lorsqu'une flotte autonome est constituée d'un seul et même type d'entités, on qualifie alors cette flotte d'*homogène*. Dans le cas contraire, *i.e.* quand la flotte contient des entités de types différents, on parle de *flotte hétérogène* [51].

Le dictionnaire Larousse définit une *flotte* comme une *réunion de navires naviguant ensemble et appartenant à un même domaine d'activité*. Par ailleurs, il définit un *essaim* comme *une troupe d'insectes de la même espèce*. Bien que ces définitions ne soient pas directement liées au domaine des systèmes autonomes, il est possible d'en faire une adaptation. La notion d'appartenance à un même groupe est commune

à ces deux définitions (*i.e.* elles utilisent les mots *réunion* et *troupe*). Cette notion de groupe fait ainsi écho au concept de *flotte* évoqué au début de la section 1.2. La principale différence entre ces deux définitions est l'uniformité. En effet, un essaim implique une homogénéité des entités qui le composent (*i.e.* la définition du terme *essaim* précise que les entités sont de la *même espèce*) tandis que la flotte implique uniquement une notion de groupe pouvant donc être constituée d'entités différentes (*i.e.* la définition du terme *flotte* fait référence à un *même domaine d'activité*).

Nous proposons donc deux nouvelles définitions pour caractériser les groupes de systèmes autonomes :

Définition 1.3. *Une flotte est un groupement d'entités mobiles.*

Définition 1.4. *Un essaim est une flotte constituée d'entités mobiles similaires*[1].

Les projets mettant en œuvre des essaims sont nombreux. Nous en donnons ici quelques exemples. Le projet *Centibots* de l'*université de Washington* utilise une centaine de robots terrestres autonomes afin de réaliser trois missions successives [59, 60]. La première consiste à explorer et à cartographier un environnement intérieur inconnu (on parle de *Simultaneous Localization and Mapping* - SLAM). La seconde mission est la localisation d'un élément précis de l'environnement. Enfin la dernière consiste à surveiller et détecter une potentielle intrusion dans une zone restreinte autour de l'élément précédemment trouvé. Dans cet exemple, les robots sont tous de même type (*i.e.* robots terrestres) et possèdent une configuration identique. De plus ils suivent tous les mêmes objectifs et, pour les atteindre, opèrent de manière identique. Nous pouvons également citer le projet *Kilobots* de l'*université d'Harvard* qui utilise, quant à lui, un millier de robots à pattes vibrantes dans le but de reproduire des formes géométriques pré-définies [61]. Ici, les robots sont également strictement identiques. Enfin, le projet *Swarmie* de la *NASA* met en œuvre un groupe homogène de robots afin de rechercher et de collecter des échantillons de matériaux particuliers sur d'autres planètes ou astres.

Les projets utilisant des flottes hétérogènes sont en revanche beaucoup plus rares et sont principalement dédiés à l'exploration. Nous pouvons par exemple citer un projet d'exploration de bâtiment fortement endommagé utilisant à la fois un robot terrestre et un drone aérien [62]. Le robot terrestre visite le maximum d'endroits possibles pour cartographier l'intérieur du bâtiment et fait appel au drone posé au-dessus de lui pour explorer les zones inaccessibles. Cet exemple démontre l'intérêt d'une flotte hétérogène pour ce type de scénario. Un autre projet d'exploration utilise quant à lui une flotte de cinq robots terrestres équipés de capteurs différents [63]. Ici, l'hétérogénéité n'est pas matérialisée par l'utilisation de systèmes autonomes de types différents mais par la différenciation de l'équipement embarqué sur chaque robot de la flotte. Enfin un projet de l'*université Libre de Bruxelles* étudie la cohésion d'un essaim de robots dont les comportements diffèrent [64]. Certains robots prédéterminés tentent de régler leur vitesse de déplacement sur celles de leurs voisins tandis que les autres robots fixent indépendamment et individuellement leur vitesse propre. Dans ce scénario, l'hétérogénéité porte sur le comportement et non sur le type ou l'équipement des engins.

1. La notion de similarité sera abordée en section 1.2.3.

1.2.3 Niveaux d'hétérogénéité

La section précédente illustre l'existence de plusieurs niveaux d'hétérogénéité. Néanmoins, il n'existe pas à notre connaissance de classification formelle permettant de déterminer si deux engins sont *similaires* comme évoqué dans la définition 1.3. Cette notion est donc subjective et doit être définie. Nous avons vu dans la section 1.2.2 que l'hétérogénéité pouvait être matérialisée par plusieurs facteurs :

- la présence de différents *types* de systèmes autonomes dans la flotte ;
- les variations sur l'*équipement* des systèmes autonomes de la flotte :
  - dans le cas de variations sur les *capteurs*, les systèmes autonomes ne peuvent pas détecter les mêmes événements ou capter les mêmes informations ;
  - dans le cas de variations sur les *actionneurs*, les systèmes autonomes ne peuvent pas effectuer les mêmes missions.
- les variations de *comportements*.

Dans la littérature, la définition du terme *hétérogène* varie souvent selon les auteurs. Ainsi, dans les exemples présentés dans la section 1.2.2, le terme hétérogène n'a pas le même sens. Nous pouvons donc en déduire que la notion d'hétérogénéité est dépendante du scénario d'utilisation sous-jacent. C'est pourquoi il est nécessaire de préciser la définition de cette notion.

Avant de définir précisément le principe d'hétérogénéité dans une flotte, nous devons au préalable introduire la notion de capacité :

Définition 1.5. *Une **capacité** représente l'ensemble des équipements et comportements embarqués au sein d'un système autonome qui lui permettent de réaliser une mission pré-définie.*

Nous définissons l'hétérogénéité comme suit :

Définition 1.6. *Deux systèmes autonomes sont **hétérogènes** dans un scénario donné si et seulement s'ils disposent de capacités différentes pour ce scénario.*

La définition de la similarité est un corollaire de la définition précédente :

Définition 1.7. *Deux systèmes autonomes sont **similaires** dans un scénario si et seulement s'ils disposent de capacités identiques pour ce scénario.*

1.3 Modèles d'interaction dans les flottes autonomes

Les flottes d'engins autonomes peuvent suivre plusieurs stratégies pour résoudre des problèmes donnés. Ces stratégies dépendent de plusieurs facteurs tels que la capacité des nœuds à se reconnaître ou à communiquer entre eux. Nous qualifions ces stratégies de *modèles d'interaction*. Bien que ces modèles existent depuis plusieurs décennies [50, 65, 66], Parker est la première à en avoir établi une classification [67, 68]. Après avoir présenté les principaux facteurs de classification utilisés par Parker, nous présentons les différents modèles d'interaction existants.

1.3.1 Facteurs d'interaction

Parker définit trois facteurs determinant le modèle d'interaction utilisé par une flotte d'entités mobiles.

Le premier est la *conscience réciproque*. Ce facteur est défini par Iocchi *et al.* comme la faculté des entités à connaître et à prendre en compte l'existence et les actions des autres entités qui composent la flotte [51]. Cette faculté suppose la présence des capteurs nécessaires à cette reconnaissance (*e.g.* caméra, système de communication) mais également l'intégration des informations ainsi acquises dans les comportements des objets mobiles. Cette *conscience* peut résulter de deux mécanismes distincts [51] :

- communications *directes* : les entités sont équipées d'un système de communication (*e.g.* radio, infrarouge) ;
- communications *indirectes* : les entités sont capables de se détecter mutuellement et d'analyser leurs comportements respectifs mais ne peuvent pas échanger d'informations de manières explicite. Un exemple de communication indirecte est le recourt à des phéromones [69]. Cette forme de communication est définie par un mécanisme biologique appelée *stigmergie* que l'on retrouve notamment chez les colonies de fourmis [54, 70]. Ce mécanisme peut prendre deux formes distinctes [51] :
  - une stygmergie est dite *active* lorsqu'un robot effectue une action dans le but de modifier l'environnement d'une autre entité capable de détecter ce changement ;
  - une stygmergie est dite *passive* lorsqu'un robot effectue une action dans le but de modifier le résultat d'une action précédente réalisée par une autre entité.

Parker considère qu'une flotte est *consciente* si et seulement si les entités qui la composent tiennent compte des actions et des intentions des autres entités [68]. Une flotte *inconsciente* pourrait donc recourir à des communications dans l'unique but d'éviter les collisions entre engins mais pas d'optimiser la réalisation d'une ou plusieurs missions.

Le second facteur est la *nature des objectifs* qui peuvent être *communs* ou *individuels*. Dans le premier cas, les entités poursuivent le même objectif et travaillent donc de concert (*e.g.* pour déplacer une structure trop lourde pour être transportée par une seule entité). Dans le second cas, les objectifs poursuivis par les entités sont différents, chaque entité travaillant donc individuellement à la réussite de sa propre mission.

Le dernier facteur spécifie les *liens de dépendance entre les objectifs*. L'existence d'un tel lien suppose que les actions effectuées par une entité, dans le but de résoudre ses objectifs propres, sont bénéfiques pour les autres membres de la flotte. Un exemple d'un tel lien est le nettoyage d'une zone par plusieurs robots. En effet, un robot ayant nettoyé un bout de la zone évite aux autres robots de nettoyer à nouveau le même endroit.

1.3.2 Modèle collectif

Le *modèle collectif* repose sur une flotte inconsciente dont les membres partagent des objectifs communs et dont les actions sont bénéfiques aux autres membres de la flotte. Ce modèle est le plus simple car il ne nécessite pas de communication directe entre les nœuds pour permettre la réalisation des missions. Ce modèle s'inspire du comportement de certaines colonies d'insectes [54].

Un exemple d'application collective est l'exploration d'un terrain à l'aide de phéromones [69]. Dans cet exemple, un essaim de drones explore une zone inconnue. Chacun des drones de cet essaim dépose une certaine quantité de phéromones sur son passage. Ce mécanisme permet d'optimiser le processus d'exploration du terrain, un drone se dirigeant dans la direction où la quantité de phéromones est la plus faible. Ce scénario exploite bien une flotte *inconsciente* car les entités ne se basent pas sur les actions ou les intentions des autres pour prendre leurs décisions. Les entités partagent un objectif commun (*i.e.* l'exploration du terrain) et leurs actions ont un impacte positif sur les autres entités (*i.e.* l'utilisation des phéromones optimise le processus d'exploration).

1.3.3 Modèle coopératif

Le *modèle coopératif* repose sur une flotte consciente dont les membres partagent des objectifs communs et dans laquelle les actions effectuées par une entité sont bénéfiques aux autres membres de la flotte. La principale différence avec le modèle collectif est la prise en compte par les entités de la flotte des intentions et actions des autres membres. Le modèle coopératif fait plus précisément référence au *modèle coopératif actif* défini par El Zoghby *et al.* [54]. Le *modèle coopératif passif* fait quant à lui référence au modèle collectif.

Le principe de coopération peut différer en fonction des sources [71], néanmoins il est admis qu'une coopération implique l'action de plusieurs entités conscientes dans le but de résoudre une mission commune [50, 51, 54, 68].

Le projet CARUS [72] est un exemple de flotte coopérative. Dans ce scénario, des drones aériens sont utilisés pour surveiller un ensemble de points d'intérêt dans un terrain délimité. Les drones se partagent dynamiquement les zones à surveiller en s'assurant que chacune de celles-ci est couverte à tout moment. Dans cet exemple, les drones communiquent à intervalle de temps régulier afin d'échanger leur état courant, la flotte est donc consciente. La surveillance constante de l'ensemble des points d'intérêt est l'unique objectif de la flotte, et la surveillance d'une zone d'intérêt par un drone permet aux autres de ne plus s'occuper de celle-ci.

1.3.4 Modèle collaboratif

Le *modèle collaboratif* est semblable au modèle coopératif à la différence que les objectifs visés sont individuels. Dans ce modèle, chaque entité poursuit des missions qui lui sont propres mais dont la réalisation est bénéfique aux autres membres de la flotte. Les entités travaillent donc seules sur des missions distinctes nécessitant

le plus souvent une spécialisation [67, 68]. Par conséquent, les flottes collaboratives sont le plus souvent hétérogènes.

Le *blackboard* est un exemple de système exploitant un modèle collaboratif [73]. Ce système permet à un ensemble d'entités spécialisées de résoudre un problème complexe. Ce problème est subdivisé en tâches individuelles qui nécessitent des compétences particulières pour être résolues. La subdivision suppose que chaque tâche est réalisable par au moins une des entités présentes dans le système. Ces tâches sont réunies sur un tableau noir (*i.e.* le *blackboard*) souvent décrit comme une mémoire partagée [74]. Une entité souhaitant résoudre une tâche la verrouille sur le *blackboard* avant de commencer son travail, prévenant ainsi les autres entités que cette tâche est en cours de résolution. Une fois résolue, la tâche est marquée comme résolue sur le *blackboard*. Les entités peuvent ajouter de nouvelles tâches sur le *blackboard* au fur et à mesure de leur avancée dans la résolution du problème global. Les tâches peuvent également être ordonnées, rendant ainsi la résolution d'une tâche de rang n impossible tant que l'ensemble des tâches de rangs inférieurs n'ont pas été résolues.

1.3.5 Modèle de coalition

Le *modèle de coalition* repose sur une flotte collaborative qui nécessite de plus le recours à des stratégies coopératives pour certaines tâches particulières. En d'autres termes, certaines tâches sont partagées et nécessitent la coopération de plusieurs entités pour être résolues.

Un des scénarios les plus courants reposant sur ce modèle est le transport de marchandises dont certaines nécessitent l'action de plusieurs entités [75, 76].

1.3.6 Modèle coordonné

Le *modèle coordonné* repose sur une flotte consciente où les entités travaillent sur des problèmes individuels et dont la résolution n'a aucun impact sur les autres entités de la flotte. Les interactions ne sont ici utiles que pour limiter la gêne potentielle que les entités pourraient se causer l'une à l'autre. Ce modèle est principalement utilisé pour partager une zone géographique et implémenter des mécanismes d'évitement de collision entre des robots [77, 78].

1.3.7 Modèle compétitif

Le *modèle compétitif* suppose que les entités (ou un sous-groupe d'entités) poursuivent des objectifs qui ont un impact négatif sur les autres (ou sur un autre sous-groupe).

Ce modèle est notamment utilisé dans les scénarios où des équipes de robots s'affrontent dans un jeux sportifs [79] comme par exemple la RobotCup [80]).

1.4 Résumé

Dans ce chapitre, nous avons présenté les différents types de *systèmes autonomes* existants ainsi que leurs différents *niveaux d'autonomie*. Nous avons défini la notion d'*indépendance* d'un système autonome et avons introduit les concepts de *flotte* et d'*essaim*. Nous avons pour cela précisé les différents *niveaux d'hétérogénéité* dans les flottes et en avons déduit une définition de la *similarité*. Nous avons également décrit les principales architectures pour les flottes et avons ainsi précisé la notion de *flotte autonome*. Enfin, nous avons présenté les différents *modèles d'interaction* possibles dans les flottes de systèmes autonomes en précisant les facteurs déterminant de ces modèles.

Nous avons mis en évidence le fait que les communications au sein de ces flottes sont un point clef relatif à l'autonomie et aux interactions des systèmes autonomes. C'est pourquoi, dans le chapitre suivant, nous aborderons les principes et les problématiques des communications sans fils dans les flottes autonomes.

CHAPITRE 2

Communication dans les flottes autonomes

Sommaire

2.1 Structures réseaux

Les systèmes autonomes étant des engins mobiles, les échanges d'information dans une flotte autonome reposent sur des communications sans fil. L'organisation du réseau au sein d'une telle flotte peut prendre plusieurs formes.

Un réseau *infra-structuré* (*managed network*) consiste en un point central par lequel passe nécessairement l'ensemble de communications émises et reçues par les engins de la flotte. Ce point central est communément appelé *point d'accès*. Cette structure réseau se rapproche de l'architecture centralisée définie dans la section 1.2.1. Elle souffre donc des mêmes faiblesses quant à la robustesse et à la surcharge de calcul au niveau du point d'accès. De plus, cette solution limite la mobilité des engins qui ne peuvent pas s'éloigner au delà d'une certaine distance du point d'accès (cette distance dépend de la technologie utilisée) au risque de perdre leurs capacités de communication.

Un réseau *maillé* (*wireless mesh network*) [81] repose sur un ensemble de points centraux appelés *relais* qui ont un rôle similaire à celui du point d'accès des réseaux infra-structurés. Les relais sont responsables de l'acheminement des communications d'une partie des engins de la flottes vers d'autres engins ou relais. Ces relais peuvent eux-mêmes éventuellement être des engins mobiles. Un réseau d'antennes téléphoniques accompagné des terminaux mobiles les exploitant est donc un réseau maillé. De même, un réseau de satellites de télécommunications fournissant un accès Internet à des terminaux mobiles peut également être considéré comme un réseau maillé où les satellites sont non seulement des relais mais aussi des entités mobiles. Ce type de réseau suit une logique hiérarchique telle que présentée dans la section 1.2.1. Il souffre donc des même faiblesses.

Enfin, un réseau *ad-hoc* n'utilise aucun point central. Les communications sont directes entre les nœuds du réseau. Cette structure réseau est inspirée de l'architecture décentralisée présentée dans la section 1.2.1. La flotte étant constituée de nœuds mobiles, le réseau ad-hoc est appelé MANet (*Mobile Ad-Hoc Network*) [82]. Un MANet est un réseau constitué d'entités mobiles (généralement dénommées nœuds) qui communiquent entre elles de manière directe et sans recourir à l'utilisation d'une infrastructure pré-existante. Les nœuds du réseau communiquent donc uniquement de proche en proche (*i.e.* avec leurs voisins suffisamment proches pour permettre un échange d'informations).

Nous complétons ainsi la définition 1.2 :

Définition 2.1. *Une flotte est* ***strictement autonome*** *si et seulement si elle ne repose que sur un ensemble de systèmes autonomes indépendants organisés selon une architecture décentralisée dont l'intégralité des communications intra-flotte sont faites au travers d'un réseau ad-hoc.*

Dans la suite de ce document, la terminologie *flotte strictement autonome* sera exprimée simplement par *flotte autonome*.

2.2 Technologies et échelles de réseaux

Dans cette section, nous présentons les différentes technologies réseaux qui peuvent permettre à plusieurs UMS de communiquer. Nous classons ces technologies en fonction de leur portée et de leurs propriétés. Notre objectif est ici de déterminer les technologies les plus pertinents dans le cadre des flottes autonomes. La littérature définit plusieurs niveaux d'échelle auxquels sont dédiés des technologies de communications adaptées [83, 84].

2.2.1 Réseaux personnels (PAN)

Un réseau personnel (*Personnal Area Network* - PAN) permet aux nœuds de communiquer à des distances allant jusqu'à quelques mètres. Ce type de réseau est particulièrement adapté aux flottes d'UMS de taille réduite.

Une majorité de PANs repose sur la technologie *Bluetooth* [85] qui permet d'établir des communications radio dont le débit est limité à 24 Mbit/s. La bande de

fréquences ISM [1] utilisée par cette technologie se situe aux alentours des 2.4 GHz. Les communications *Bluetooth* ne nécessitent pas d'infrastructure, elle sont donc particulièrement adaptées aux réseaux ad-hoc. Certains UMS de tailles réduites, principalement dédiés aux loisirs utilisent cette technologie de communication (*e.g.* *Sphero* d'*Orbotix* [32], *Rolling Spider* de *Parrot* [37]).

Le *ZigBee* [86] est une autre technologie de communication pour les PANs exploitant également la bande de fréquences ISM des 2.4 GHz. Cette technologie permet de communiquer jusqu'à une limite de 250 kbit/s sur une distance d'environ 100 mètres. Le *ZigBee* est également capable d'opérer sur des bandes de fréquences plus basses (*e.g.* 868 MHz, 915 MHz) afin d'augmenter la portée des communications au détriment du débit (resp. 20 et 40 kbit/s). Consommant peu d'énergie, le *ZigBee* est utilisé principalement pour la domotique [87] mais également pour certains UMS évoluant en flotte [46, 47, 62, 88].

Le *NFC* (*Near Field Communication*) est une technologie de communication à très courte portée basée sur l'architecture RFID (*Radio Frequency Identification*) [89, 90]. Le *NFC* utilise très peu d'énergie mais possède un débit réduit (*i.e.* quelques centaines de kbit/s) pour une portée de quelques centimètres. De par ses contraintes, le NFC n'est pas utilisé comme technologie de communication pour les UMS.

L'infrarouge est une autre technologie de communication utilisée dans les réseaux de type PAN [91, 92]. Les communications infrarouges reposent sur des ondes lumineuses plutôt que sur des ondes radio. L'avantage principal de cette technologie est sa faible consommation énergétique en comparaison de celles basées sur des radio-fréquences. De plus, les ondes infrarouges permettent à la fois d'effectuer des communications mais aussi de détecter des obstacles [93]. En revanche, les communications infrarouges présentent un certain nombre de désavantages [33, 93, 94] :

— elle sont extrêmement sensibles à la luminosité ambiante ;
— elles sont fortement directionnelles ;
— elles ont un débit limité (*i.e.* quelques dizaines de kbit/s).

De par ses contraintes, l'infrarouge est rarement utilisé pour permettre à des UMS de communiquer mais plutôt comme un moyen de détecter des obstacles.

2.2.2 Réseaux locaux (LAN)

Les réseaux locaux (*Local Area Network* - LAN) permettent aux nœuds de communiquer à des distances pouvant atteindre une centaine de mètres. Ce type de réseau est compatible avec la plupart des flottes d'UMS.

La technologie principale utilisée dans les réseaux locaux sans fil est le *Wi-Fi* [95]. Elle permet d'effectuer des communications radio d'une portée d'environ 100 mètres pour un débit de plusieurs centaines de Mbit/s. Les différentes normes du *Wi-Fi* exploitent les bandes de fréquences ISM de 2.4 GHz et 5.8 GHz. Cette dernière permet d'obtenir des débits plus importants (théoriquement de l'ordre du Gbit/s) au détriment de la portée des communications (réduite à quelques dizaines de mètres). Le *Wi-Fi* peut être configuré selon plusieurs modes de fonctionnement : il est aussi bien utilisable en mode infra-structuré qu'en mode ad-hoc. Avec la démocratisation

1. Bandes de fréquences industrielles, scientifiques et médicales.

des *smart-phones*, cette technologie s'est généralisée aussi bien pour les UMS de loisirs que pour les modèles professionnels [18, 21, 28, 96].

Plus généralement, la démocratisation récente des objets connectés et de la domotique ont permis au *Bluetooth*, au *ZigBee* et au *Wi-Fi* de devenir des standards de communication pour les PANs et les LANs [97].

2.2.3 Autres échelles de réseaux

Afin de compléter la liste des échelles de réseaux existantes, nous présentons ici les échelles de tailles supérieures (MAN/RAN/WAN) et inférieures (BAN). Ces échelles sortent néanmoins du cadre de notre étude et ne seront pas considérées dans la suite de ce document.

Un *MAN* (*Metropolitan Area Network*) est un réseau dont la couverture s'étend sur une zone dont la taille est comparable à celle d'une agglomération (*i.e.* plusieurs kilomètres). Le *WiMAX* [98] est certainement la technologie la plus courante pour ce type de réseau. il permet d'atteindre des débits d'une centaine de Mbit/s pour une portée de quelques dizaines de kilomètres [84]. Un réseau WiMAX utilise un ensemble de points d'accès fixes communiquant par des ondes hertziennes. Les réseaux d'opérateurs mobiles (*i.e.* 2g, 3g, 4g, etc.) peuvent également être considérés comme des MANs exploitant un ensemble d'antennes relais formant un réseau maillé [99]. Dans ces deux cas, les nœuds mobiles sont des clients et doivent être rattachés à un point d'accès pour transmettre et recevoir des données. Un *RAN* (*Regional Area Network*) est un réseau dont la couverture s'étend sur une distance comparable à celle d'une région (*i.e.* une centaine de kilomètres) [100]. Ce type de réseau repose sur des ondes à hautes fréquences exploitant à la fois les bandes de la télévision et des réseaux d'opérateurs mobiles [101]. Ce type de réseau nécessite l'utilisation de mécanismes d'allocation dynamique de bandes de fréquences tel que la radio cognitive. Enfin, un *WAN* (*Wide Area Network*) est un réseau de taille supérieure (*i.e.* plus de 100 km de couverture) reposant généralement sur des points d'accès situés à haute altitude. Les *HPA* (*High Altitude Platforms*) sont des dispositifs situés au niveau de la stratosphère (*i.e.* altitude d'une vingtaine de kilomètres) qui permettent de former un WAN. Les projets *Loon* de *Google* [102] et *internet.org* de *Facebook* [103] sont des exemples de WAN reposant sur des HPAs (aerostats pour Google et drones à voilures fixes pour Facebook). Un WAN peut également reposer sur un ensemble de satellites évoluant bien au dessus de la couche d'ozone (*i.e.* altitude de plusieurs centaines de kilomètres). Les systèmes de positionnements tels que GPS, GLONASS ou encore GALILEO utilisent ce type d'infrastructure [104]. Ces réseaux de grande taille reposent nécessairement sur une infrastructure créant ainsi des réseaux infra-structurés ou maillés. Ce type de réseau n'est donc pas adapté à notre champ d'étude.

Un réseau corporel (*Body Area Network* - BAN) est dédié aux flottes d'entités de tailles extrêmement réduites (*i.e.* de l'ordre du millimètre) évoluant sur ou dans le corps humain [105, 106]. Bien que ces réseaux forment une structure ad-hoc, ils possèdent de nombreuses contraintes spécifiques qui sortent du cadre de notre étude.

2.2.4 Conclusion

Comme nous l'avons décrit dans les sections 1.2.1 et 2.1, les flottes strictement autonomes forment une structure ad-hoc reposant sur une architecture décentralisée. Les PANs et les LANs sont donc les échelles les plus appropriées à notre problématique. Plus particulièrement, nous avons identifié le *Bluetooth*, le *ZigBee* ainsi que le *Wi-Fi* comme les technologies de communication les plus pertinentes pour permettre à plusieurs UMS de communiquer.

2.3 Diffusion de l'information

Une flotte autonome est, par nature, en constante évolution. En effet, les entités qui la composent se déplacent dans l'espace et communiquent de proche en proche comme nous l'avons vu dans le chapitre 1 et dans la section 2.1. Deux entités dont l'éloignement est supérieur à la portée de leur système de communication ne peuvent donc pas directement échanger d'informations. Il est néanmoins possible d'utiliser un ensemble de nœuds dits intermédiaires pour relayer l'information de son émetteur vers son destinataire ; on parle de réseau *multi-sauts*. Pour qu'une information soit correctement acheminée dans un réseau de ce type, il existe principalement deux techniques : le *routage* et l'*inondation*. Notre objectif est ici de comparer ces deux techniques en mettant en évidence leurs atouts ainsi que leurs désavantages respectifs, notamment au regard des propriétés du réseau sur lequel elles sont utilisées. Pour cela, nous détaillons tout d'abord l'ensemble des déclinaisons de chacune de ces deux techniques de diffusion.

2.3.1 Routage

Le routage consiste à déterminer un chemin (*i.e.* une route) entre deux nœuds du réseau afin d'acheminer correctement des messages de l'un à l'autre. Ce mécanisme permet de faire communiquer des nœuds qui sont trop éloignés pour pouvoir communiquer directement. Dans le cas des MANets, les nœuds se déplacent au cours du temps modifiant ainsi la topologie réseau sous-jacente. Cette mobilité implique donc une instabilité des routes qui doivent être re-calculées plus ou moins souvent en fonction de la dynamique du réseau. Plusieurs familles de protocole de routage existent dans les MANets [107, 108, 109, 110, 111]. Nous présentons ici les plus courantes accompagnées de quelques exemples représentatifs.

Protocoles proactifs

Un protocole *proactif* a pour objectif de maintenir en permanence des routes entre tous les nœuds du réseau. Pour cela, ce type de protocole utilise des tables de routage locales à chaque nœud qui permettent d'établir la liste des relais auxquels il faut envoyer une information pour que celle-ci soit transmise à un destinataire précis. Cette stratégie implique des échanges réguliers entre les nœuds du réseau afin d'établir et de maintenir les routes ainsi calculées. Généralement, la totalité (ou

un sous-ensemble) des tables de routage sont échangées entre les nœuds voisins afin d'établir et de maintenir des routes cohérentes. Les protocoles proactifs utilisent donc une part substantielle et permanente de bande passante disponible. Les messages relatifs à ces protocoles qui permettent d'établir et de maintenir les routes sont appelés *messages de contrôle*.

DSDV (*Destination Sequenced Distance Vector*) [112] est un des premiers protocoles proactifs a avoir été proposé pour les MANets. Dans ce protocole, les nœuds transmettent la totalité de leur table de routage à leurs voisins permettant ainsi la construction et le maintien de tables de routage globales. Ces tables contiennent l'ensemble des destinataires possibles (*i.e.* tous les nœuds connus) avec indication du relais le plus pertinent (nombre minimum de sauts dans ce cas) pour pouvoir établir une communication. Les mises à jour des tables de routage peuvent être effectuées de manière incrémentale (*i.e.* en ne transmettant que les nouvelles entrées ou celles qui ont été modifiées) afin de réduire la quantité de bande passante (*i.e.* la taille et le nombre des messages de contrôle) nécessaire au maintien des routes.

GSR (*Global State Routing*) [113] est une amélioration du protocole DSDV. GSR maintien plusieurs tables dont une table de voisinage qui lui permet de cibler les destinataires des messages incrémentaux utilisés pour maintenir la cohérence des routes ainsi calculées. GSR réduit sensiblement la quantité de bande passante utilisée en comparaison de DSDV dans la mesure où il utilise des messages de taille inférieure et en nombre plus limité.

FSR (*Fisheye State Routing*) [114] repose sur le même principe que GSR mais adapte les informations de mise à jour en fonction de la distance (*i.e.* du nombre de sauts) entre les nœuds. En effet, pour chaque nœud du réseau, les messages de contrôle relatifs aux nœuds les plus proches sont échangés plus régulièrement que ceux qui concernent les nœuds plus éloignés. Cela permet principalement de réduire la taille moyenne des messages de contrôle au détriment de la fréquence d'actualisation des informations relatives aux nœuds les plus éloignés.

OLSR (*Optimised Link State Routing*) [115, 116] est un autre protocole proactif dont la seconde version a été standardisée en 2014. Dans ce protocole, la table de routage maintenue dans chacun des nœuds ne contient que les destinataires qui sont situés à un nombre limité de sauts s. OLSR repose parallèlement sur un ensemble de nœuds particuliers, sélectionnés de manière distribuée, appelés relais multi-points qui ont la responsabilité de relayer les messages provenant des nœud sources qui sont situés à plus de s sauts de leur destination. Les nœuds ne maintiennent donc que des tables de routage réduites à leur voisinage à s sauts. Cela permet de réduire la taille des tables et des messages de contrôle.

Protocoles réactifs

Les protocoles *réactifs* construisent les routes au moment où les informations doivent être transmises. En d'autres termes, les routes sont calculées à la demande et non continuellement comme c'est le cas pour les protocoles proactifs. La stratégie réactive ne nécessite pas d'échanges à intervalles réguliers entre les nœuds. Le principal inconvénient de ce type de protocole est le délai important nécessaire à l'établissement des routes. En effet, contrairement au modèle proactif, l'établisse-

ment d'une route réactive nécessite un envoi de messages vers la destination ainsi que la réception (et donc l'attente) d'une réponse de cette dernière.

DSR (*Dynamic Source Routing*) [117] est un des premiers protocoles réactifs développés pour les MANets. Lorsqu'un nœud s souhaite communiquer avec un nœud t, un message de demande de route est émis par s et est relayé par *broadcast* de proche en proche jusqu'à atteindre le nœud t. Ce dernier émet un message de réponse contenant la liste des nœuds par lesquels le message de demande est passé. Le message de réponse est relayé jusqu'au nœud s. Dès lors, le nœud s peut transmettre des informations à t en empruntant la route ainsi découverte. On peut noter que l'établissement d'une route par le protocole DSR nécessite un échange bidirectionnel (*i.e.* demande et réponse) entre la source et la destination.

AODV (*Ad hoc On-demand Distance Vector*) [118, 119] est une amélioration de DSR. Dans AODV, les nœuds possèdent des tables de routage qui sont mises à jour lors du passage des messages de demande de route. Ainsi, ils sont capables d'établir certaines communications sans avoir besoin de rejouer le processus de découverte de route. Cette amélioration a pour principal objectif de réduire le nombre ainsi que la taille moyenne des messages de contrôle.

TORA (*Temporally Ordered Routing Algorithm*) [120] est un autre exemple de protocole réactif qui stocke dans ses tables de routage (construites de manière opportuniste comme dans le cas d'AODV) plusieurs chemins pour une même destination. Cela permet notamment de réduire l'effet des changements topologiques qui ont lieu dans le réseau. La principale conséquence est une diminution du nombre de messages de contrôle.

Protocoles hybrides

Les protocoles *hybrides* reposent sur l'utilisation conjointe de stratégies réactives et proactives. Ces protocoles définissent un ensemble de zones dans lesquels un routage proactif est opéré. Les échanges entre les nœuds appartenant à des zones différentes sont quant à eux effectués par l'intermédiaire d'une méthode réactive. Les performances de ce type de protocoles sont fortement liées aux tailles des zones prédéfinies [121]. Le modèle hybride est adapté aux réseaux de dimension importante [109].

ZRP (*Zone Routing Protocol*) [122] est un des exemples les plus simples de protocoles hybrides. Ce protocole définit une zone comme un ensemble de nœuds dont la distance deux à deux ne dépasse pas une limite fixée. Les distances sont établies de manière logique en nombre de sauts.

ZHLS (*Zone Based Hierarchical Link State*) [123] utilise quant à lui les positions géographiques des nœuds (et donc la distance physique entre les nœuds) pour définir les zones. Ce protocole nécessite donc l'utilisation d'un système de positionnement global (*e.g.* GPS).

Protocoles géographiques

Le modèle *géographique* utilise la position des nœuds pour optimiser l'établissement des routes. Cette position est généralement obtenue par un système de po-

sitionnement globale (*e.g.* GPS) mais peut être également obtenue par d'autres indicateurs comme la puissance du signal [110, 124]. Les protocoles de routage géographique reposent généralement sur des protocoles d'autres familles (*i.e.* réactifs, proactifs, hybrides) mais utilisent en plus des données géographiques pour améliorer le processus d'établissement des routes ou la qualité des routes existantes [107, 125].

ZHLS présenté dans le paragraphe précédent est un exemple de protocole géographique reposant sur une stratégie hybride.

2.3.2 Inondation

Les techniques de routage présentées dans la section précédente permettent à une source d'obtenir un ou plusieurs chemins vers une destination avant de commencer à lui transmettre des informations. L'*inondation*, appelée également *diffusion épidémique* [126], consiste quant à elle à diffuser directement l'information dans le réseau sans nécessiter de chemin. Lorsqu'une source s souhaite communiquer avec un destinataire t, celle-ci envoie l'information à tous ses voisins (ou à un sous-ensemble de ses voisins) qui eux-mêmes re-diffusent l'information en espérant atteindre t. L'inondation est donc caractérisée par l'absence d'établissement de route.

Cette technique peut saturer le réseau en raison des nombreuses rediffusions qui doivent être opérées par l'ensemble des nœuds intermédiaires qui relaient le message provenant de s et à destination de t. Ce phénomène, qui peut aboutir à une impossibilité totale de communication, s'appelle une *tempête de broadcast* [127]. Cette tempête est due à deux phénomènes distincts que sont les *collisions* et la *contention*. Les ondes radio se chevauchent lorsqu'elles se situent dans une zone géographique trop proche. Ce chevauchement peut alors compromettre l'intégrité des informations transportées ; on appelle ce phénomène physique *collision*. La plupart des technologies de communication sans fil utilisées dans les MANets sont équipées de systèmes d'évitement de ces collisions [83, 128]. Ces systèmes reposent la plupart du temps sur un mécanisme d'accès concurrent dont l'objectif est d'interdire à des nœuds voisins d'émettre en même temps. De plus, en cas de trafic intensif et continu, les nœuds peuvent être dans l'impossibilité d'émettre de nouveaux messages. Cette saturation du réseau est un effet de bord inhérent aux mécanismes d'évitement de collision utilisés ; on appelle cette saturation *contention*. On parle de *tempête de broadcast* lorsque le nombre de collisions ou le taux de contention empêchent les nœuds de communiquer entre eux.

Il existe néanmoins plusieurs méthodes permettant de réduire l'impact de ce phénomène [127, 129] ; nous donnons ici quelques exemples de ces mécanismes.

Une des premières techniques consiste à désynchroniser les envois de messages pour réduire la probabilité de collision. Cette désynchronisation s'effectue en ajoutant un délai d'attente dont la durée est aléatoire (mais bornée) avant l'émission d'un message. Ce délai est appelé *gigue* (*jitter*) [130]. Cette technique ne permet néanmoins pas de réduire le nombre de messages émis sur le réseau mais seulement de minimiser les probabilités de collision au prix de délais d'acheminement plus importants [131].

L'*agrégation de messages* permet à un nœud intermédiaire de fusionner plusieurs messages reçus pour n'en former plus qu'un [132]. Ainsi, les messages transmis seront

moins nombreux mais de taille plus importante.

L'*approche par compteur* [133] consiste à limiter l'émission d'un message reçu ou envoyé plus de n fois pour une durée prédéfinie. Le choix de cette limite n a un impact significatif sur l'efficacité de cette solution [127, 129, 133].

L'*approche probabiliste* [134] repose sur un mécanisme de tirage pseudo-aléatoire pour déterminer si un message doit être rediffusé. Le nombre de voisins ainsi que la valeur de la probabilité choisie influent sur l'efficacité de cette solution [127, 134]. En effet, une probabilité trop faible conduirait un message à n'être que trop peu diffusé tandis qu'une probabilité trop importante pourrait n'avoir qu'une protection négligeable face à une *tempête de broadcast*. Ce phénomène est d'autant plus vrai que le nombre moyen de voisins par nœud est important.

L'*approche par distance* utilise l'espacement entre les nœuds pour déterminer si un message doit être retransmis. Cette technique est parfois couplée à des mécanismes d'adaptation de puissance de transmission dans le but d'optimiser également la consommation énergétique [135]. Cette technique nécessite l'utilisation d'un système de positionnement global ou bien la mesure de la puissance des signaux des messages reçus. Même si le choix des seuils de distance pour lesquels les messages sont retransmis ont un impact sur les performances de l'approche par distance, ce paramètre n'est pas aussi critique que ceux des approches probabiliste et par compteur [127].

L'*approche par connaissance du voisinage* consiste à prendre en compte le voisinage de l'émetteur à quelques sauts (généralement 1 ou 2) pour optimiser la diffusion des messages [136]. Cette technique nécessite d'ajouter aux paquets des informations supplémentaires sur le voisinage du nœud émetteur.

L'*approche par priorité* consiste à contrôler la diffusion des messages en fonction de leur urgence [137]. La répartition des priorités entre les messages a un impact significatif sur les performances de cette solution. En effet, si les messages sont tous de priorité maximale, l'approche par priorité n'aura qu'un impact négligeable.

2.3.3 Discussion

L'inondation et le routage constituent deux méthodes pour relayer des informations au travers des réseaux MANets. Le choix de l'une ou l'autre de ces deux techniques dépendent de plusieurs facteurs que nous détaillons dans cette section. Ces facteurs ont un impact plus ou moins important sur un ensemble de métriques ; celles auxquelles nous nous intéressons sont les suivantes :

- Le *taux de succès* qui est un ratio entre le nombre de paquets émis et le nombre de paquets reçus. La différence entre ces deux valeurs est principalement due aux collisions et aux contentions qui peuvent entraîner des pertes de paquets ainsi qu'aux fautes de routage lorsqu'aucune route n'est trouvée.
- Le *taux d'utilisation* qui mesure la quantité de bande passante utilisée par la méthode choisie. Cette métrique peut avoir un impact plus ou moins important sur le *taux de délivrance*. Le taux d'utilisation a également un impact sur la consommation énergétique des nœuds.
- Le *délai de communication* qui évalue le temps moyen nécessaire à une donnée pour être transmise d'une source vers sa destination.

Taille du réseau

La taille du réseau (*i.e.* son nombre de nœuds) est un des premiers facteurs à prendre en compte lors du choix d'une méthode de communication.

Dans le cas du routage, les différentes études semblent indiquer que les protocoles hybrides sont les plus efficaces sur les réseaux de taille importante (*i.e.* au delà de plusieurs centaines de nœuds) [109]. En revanche, les délais peuvent être fortement impactés lors des communications entre des nœuds n'appartenant pas aux mêmes zones. L'augmentation du nombre de nœuds dans le réseau implique une majoration des délais dans l'établissement des routes pour les protocoles réactifs et un accroissement du taux d'utilisation dans le cas des protocoles proactifs [138]. Pour un réseau de taille modéré, la seule prise en compte de la taille du réseau ne semble pas déterminante dans le choix d'une stratégie de routage particulière [139, 140].

L'inondation est quant à elle plus sensible à la taille du réseau pour ce qui est du taux de succès et du taux d'utilisation [129, 141]. En revanche, les délais ne semblent pas impactés de manière significative par la taille du réseau [141]. Enfin, l'utilisation de techniques d'optimisation, comme celles présentées dans la section 2.3.2, permettent de réduire substantiellement l'effet de la taille du réseau sur ces métriques [129].

Densité du réseau

La densité du réseau intègre le nombre moyen de voisins par nœud. Une forte densité assure une bonne connectivité mais peut conduire à une saturation des communications en cas de trafic important. Couplée à la taille du réseau, la densité est un facteur à prendre en compte lors du choix d'une méthode de communication.

Peu d'études sur les protocoles de routage se sont focalisées sur la densité, préférant plutôt s'intéresser à la taille du réseau comme nous l'avons vu précédemment. Il a néanmoins été démontré qu'un nombre important de voisins implique une augmentation du taux d'utilisation pour les protocoles réactifs et peut donc conduire à une saturation du réseau dans le cas d'un trafic soutenu [142]. Dans le cas des protocoles proactifs, la densité n'aura d'effet significatif sur le le taux d'utilisation que si la dynamique du réseau est importante, le volume des messages de contrôle étant principalement dépendant des changements topologiques [143]. En revanche, une densité trop faible conduit à une impossibilité d'établir des routes et rend donc impossible l'utilisation des protocoles réactifs ou proactifs pour des réseaux peu connectés [144]. L'impact de la densité sur les protocoles hybrides dépendra de la nature des communications (*i.e.* intra-zone ou inter-zone) [121]. En effet, de par leur utilisation conjointe de stratégies réactives et proactives, les protocoles hybrides combinent les avantages mais aussi les inconvénients de ces deux familles de protocoles [110].

L'inondation basique est également sensible à la densité du réseau. Une forte densité accompagnée d'un trafic important auront un impacte négatif sur les performances de cette stratégie [126, 134, 136]. L'utilisation d'optimisations permet ici encore de réduire l'impact de la densité du réseau [127, 129, 136]. En revanche, l'inondation ne nécessitant pas l'établissement de routes préalables, une faible densité et une forte dynamique n'empêcheront pas une donnée d'être transmise.

Trafic réseau

L'importance du trafic réseau est un paramètre qui peut avoir plus ou moins d'impact sur les stratégies de communication.

Le nombre de messages de contrôle échangés dans les protocoles proactifs ne dépend pas du volume du trafic. Les délais, le taux d'utilisation ainsi que le taux de succès restent donc stables en présence d'un trafic réseau soutenu [138]. En revanche, le trafic peut devenir assez important pour saturer les capacités de la technologie de communication utilisée, on parle alors de *congestion*. Les protocoles réactifs sont quant à eux beaucoup plus sensibles au volume du trafic réseau. En effet, ces protocoles font circuler des messages de contrôle lors de l'envoi d'informations et non à intervalles de temps réguliers. La quantité de messages de contrôle est donc dépendante du trafic réseau. Par conséquent, une augmentation des délais ainsi que du taux d'utilisation accompagnée d'une chute du taux de succès sont observés lors d'une augmentation de trafic [138]. Dans le cas des protocoles hybrides, les pertes de performance auront lieux dans les échanges inter-zones (*i.e.*, les échanges reposant sur une stratégie réactive).

L'inondation n'est pas adaptée à un trafic important [145]. En effet, le taux d'utilisation de cette stratégie est par nature élevé. Or, même avec l'aide des optimisations présentées en section 2.3.2, les messages sont répliqués et échangés plusieurs fois dans le réseau augmentant ainsi le risque de congestion [129]. Ainsi, une augmentation du trafic engendrera une augmentation du taux d'utilisation et donc une diminution du taux de succès accompagnée d'une augmentation des délais. Le volume du trafic réseau est une des principales causes de la *tempête de broadcast* [127].

Volatilité des nœuds

Un réseau est dit volatil lorsque n'importe lequel de ses nœuds peut disparaître ou apparaître à tout moment sans en informer les autres. Ce phénomène peut se produire en cas de panne d'un nœud ou en présence d'une mobilité particulière engendrant des déconnexions longues de certaines entités. La volatilité est un phénomène qui ne peut pas être ignoré dans le cadre des flottes autonomes. En effet, une entité peut rencontrer une panne à tout moment sans pour autant pouvoir prévenir les autres membres de la flotte [146].

Les protocoles proactifs sont particulièrement sensibles à la volatilité du réseau [144]. En effet, la perte ou l'apparition d'un nœud entraînent nécessairement des mises à jour de certaines routes précédemment établies et augmentent ainsi en cas de volatilité importante le taux d'utilisation au détriment des performances. Les protocoles réactifs sont moins sensibles à ce phénomène sauf lorsque la volatilité est telle qu'une route est impossible à établir [144]. Reposant à la fois sur les deux stratégies, les protocoles hybrides sont donc soumis aux mêmes faiblesses.

L'inondation, de par son fonctionnement qui consiste à répliquer les messages sans pour autant nécessiter l'existence d'un chemin de bout en bout pour amorcer la diffusion, est tolérante à une volatilité importante.

Mobilité des nœuds

La mobilité des nœuds engendre des changements topologiques plus ou moins importants, influant ainsi sur la connectivité du réseau [141, 147]. Ces changements de connectivité peuvent être plus ou moins nombreux et fréquents en fonction du modèle de mobilité utilisé [2] [148, 149, 150]. De plus, le changement de certains paramètres au sein d'un même modèle de mobilité peut également avoir un impact sur la connectivité du réseau et donc sur les performances des stratégies de communication mises en œuvre [140, 144, 147]. De plus, l'existence d'un chemin de bout en bout n'est pas toujours avérée [144]. Pour ces raisons, les protocoles de routage sont fortement impactés par la mobilité du réseau.

L'approche par inondation, bien que tributaire de la connectivité du réseau, tolère les changements topologiques importants et donc une forte mobilité des nœuds [129, 151, 152, 153].

Présence d'obstacles

Il est clair que la présence d'obstacles peut modifier la connectivité du réseau de manière plus ou moins significative [154, 155, 156] tout comme la mobilité ou la volatilité des nœuds, Par conséquent, les conclusions émises dans les paragraphes traitant de la volatilité et de la mobilité sont également applicables dans les scénarios mobiles intégrant des obstacles dans l'environnement.

Conclusion

Dans cette section, nous avons discuté d'un certain nombre de facteurs pouvant affecter les performances des différentes stratégies de communication. L'analyse que nous avons faite au travers de différents facteurs et métriques permet d'affirmer qu'aucune stratégie de communication n'est optimale dans toutes les situations. Par conséquent, le choix d'une de ces stratégies est *contextuel*. Notre discussion a permis de mettre en avant les éléments suivants qui doivent être considérés dans le choix d'une stratégie de diffusion :

— L'*échelle du réseau* : la densité ainsi que la taille du réseau ont un impact significatif sur les performances des différentes stratégies de communication. En effet, une taille importante ou une densité forte peuvent engendrer des latences dans les communications. En revanche, une densité trop faible peut faire obstacle à la formation de routes valides.

— Le *volume du trafic* : la quantité ainsi que la taille des messages échangés par les nœuds doivent également être prises en compte. En effet, un volume important de trafic peut conduire à une saturation des communications.

— Les *changements topologiques* : les modifications dans la connectivité du réseau peuvent avoir un impact important voire entraver le fonctionnement de certaines stratégies de communication.

2. Les modèles de mobilité ainsi que leurs impacts respectifs sur les flottes seront abordés dans le chapitre 3.

2.4 Résumé

Dans ce chapitre, nous avons présenté les différentes *architectures réseau* existantes pour les réseaux mobiles et avons défini la notion de *flotte strictement autonome*. Nous avons également détaillé les différentes *échelles de réseaux* et les *technologies de communication* associées. Enfin, nous avons abordé la problématique de la *diffusion de l'information* dans les réseaux mobiles ; nous avons pour cela développé deux stratégies de communication que sont le *routage* et l'*inondation*. Après avoir détaillé ces stratégies, nous les avons comparées au regard de plusieurs facteurs (taille du réseau, densité, trafic, volatilité, mobilité et présence d'obstacles) et de plusieurs métriques (taux de succès, taux d'utilisation et délais de communication). Nous avons conclu de cette analyse que *le choix d'une stratégie de communication est dépendant du contexte* (*i.e.* du scénario d'application) et avons mis en exergue trois critères significatifs permettant un aiguillage dans le choix d'une de ces stratégies : l'échelle du réseau, le volume du trafic ainsi que les changements topologiques.

Nous avons pu déterminer au travers de la section 2.3 que la *connectivité* du réseau est un élément décisif dans les performances des stratégies de communication. Le *modèle de mobilité* des nœuds est un des principaux facteurs ayant un impact significatif sur cette connectivité. Fort de ces résultats, nous nous intéresserons dans le chapitre suivant à différents modèles de mobilité adaptés aux systèmes autonomes et à leur impact sur la connectivité du réseau sous-jacent.

CHAPITRE 3

Modèles de mobilité

Sommaire

3.1 Introduction

Comme nous l'avons constaté dans le chapitre précédent, la mobilité des nœuds est un facteur décisif dans le choix d'une stratégie de communication au sein d'une flotte autonome. L'étude de Tracy Camp *et al.* [147] sur les modèles de mobilité les plus courants dans les réseaux de type MANets met en lumière les éléments à prendre en compte lors de l'évaluation de protocoles réseaux :

— les modèles de mobilité influent de manière significative sur les performances des protocoles réseaux ;
— l'évaluation d'un protocole réseau devrait être effectuée en utilisant un modèle de mobilité *réaliste* pour le scénario visé ;
— dans le cas où de telles traces seraient indisponibles, le protocole devrait être évalué avec plusieurs modèles de mobilité.

Fort de cette analyse, nous présentons dans ce chapitre les modèles de mobilité les plus courants pour les flottes autonomes. Pour chacun de ces modèles, nous détaillons son fonctionnement et présentons ses propriétés. Nous nous intéressons plus particulièrement à la répartition géographique des entités dans une zone prédéfinie ainsi qu'à la connectivité entre les nœuds. Enfin, nous discutons de son réalisme dans des conditions réelles.

3.2 Modèles de mobilité généralistes

Un modèle de mobilité *généraliste* est un modèle qui n'est pas spécifique à un type d'engin particulier. En d'autres termes, il ne prend pas en compte les potentielles contraintes de mouvement que peuvent avoir les engins réels. Nous présentons dans cette section les modèles généralistes les plus courants que l'on peut trouver dans la littérature.

3.2.1 Random Waypoint

Le *Random Waypoint* (RWP) est certainement le modèle de mobilité le plus utilisé dans les simulations sur les réseaux de type MANets [147, 157, 158]. Une entité utilisant ce modèle se déplace de manière indépendante des autres en changeant de direction et de vitesse de déplacement selon une stratégie aléatoire. L'algorithme 3.1 décrit le fonctionnement de ce modèle que nous illustrons par ailleurs à la figure 3.1.

Algorithm 3.1 Random Waypoint

Require: $maxSpeed, maxSleepTime, area$
  repeat
    $target \leftarrow randomTargetPointInArea(area)$
    $speed \leftarrow randomSpeed(maxSpeed)$
    $goToTargetWithSpeed(target, speed)$
    $sleepTime \leftarrow randomTime(maxSleepTime)$
    $sleep(sleepTime)$
  end repeat

Différentes études montrent que ce modèle tend à créer une concentration des nœuds dans la partie centrale de la zone prédéfinie [147, 150, 157, 158, 159, 160]. Par conséquent, la connectivité est meilleure dans cette partie qu'en bordure de zone où la connectivité est beaucoup plus faible. Par ailleurs, certaines expérimentations ont montré que la connectivité moyenne sur ce modèle varie au cours du temps (*i.e.* on constate une alternance entre croissance et décroissance). Le *Random Waypoint* ne permet donc pas d'obtenir une répartition uniforme des nœuds dans la zone prédéfinie. Les paramètres internes (*i.e.* la vitesse maximale des entités et la durée maximale de pause) ont un impact sur la connectivité induite par ce modèle [147, 161].

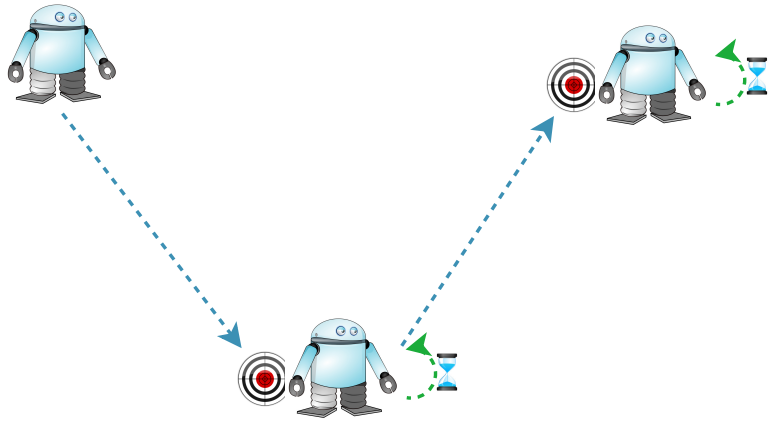


FIGURE 3.1 – Illustration du modèle de mobilité *Random Waypoint*.

Le *Random Waypoint* est réaliste pour les véhicules terrestres capables d'effectuer des rotations angulaires importantes. En effet, la direction choisie étant aléatoire, il est possible que l'engin doive effectuer des rotations complètes sur lui même. Ce type de mouvement est impossible à réaliser pour certains engins terrestres comme ceux reposant sur un système à chenilles. Ce modèle n'est pas non plus adapté pour une grande partie des engins aériens [149, 150]. En effet, les contraintes aérodynamiques de ces appareils ne leur permettent pas d'effectuer des virages d'angle trop serrés qui risqueraient de provoquer un décrochage.

3.2.2 Random Walk

Le *Random Walk* (RW) est lui aussi un modèle de mobilité reposant sur une stratégie aléatoire [147, 162]. Son fonctionnement est décrit par l'algorithme 3.2 (les comportements en bordure de zone ne sont pas représentés dans cette figure) et illustré à la figure 3.2.

Algorithm 3.2 Random Walk

Require: $maxSpeed, maxTravelTime, maxSleepTime, area$
  repeat
    $direction \leftarrow randomDirection()$
    $speed \leftarrow randomSpeed(maxSpeed)$
    $time \leftarrow randomTime(maxTravelTime)$
    $goToDirectionWithSpeedForTime(direction, speed, time, area)$
    $sleepTime \leftarrow randomTime(maxSleepTime)$
    $sleep(sleepTime)$
  end repeat

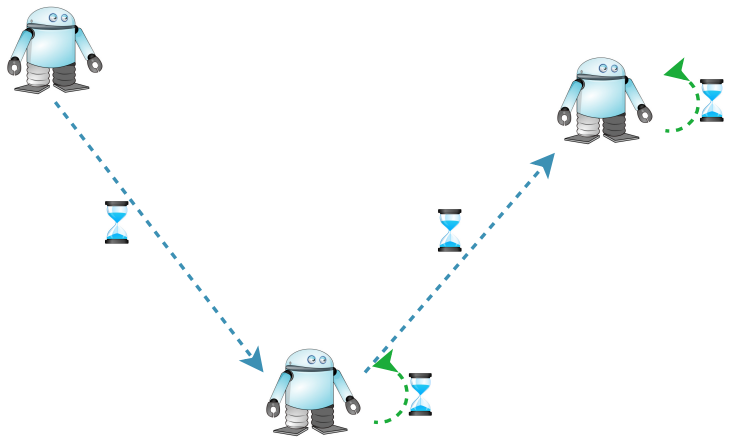


FIGURE 3.2 – Illustration du modèle de mobilité *Random Walk*.

La principale différence entre le *Random Walk* et le *Random Waypoint* est la condition d'arrêt de l'engin. Le *Random Walk* utilise une durée aléatoire tandis que le *Random Waypoint* utilise les coordonnées d'un point de destination choisi aléatoirement dans la zone. Les effets des paramètres du modèle (*i.e.* la vitesse maximale des entités et la durée maximale de pause) ainsi que la concentration des nœuds observée avec le *Random Waypoint* se retrouvent également avec le *Random Walk* [147, 150]. Néanmoins, cette concentration est plus ou moins marquée en fonction de la borne fixée pour la durée de déplacement [147].

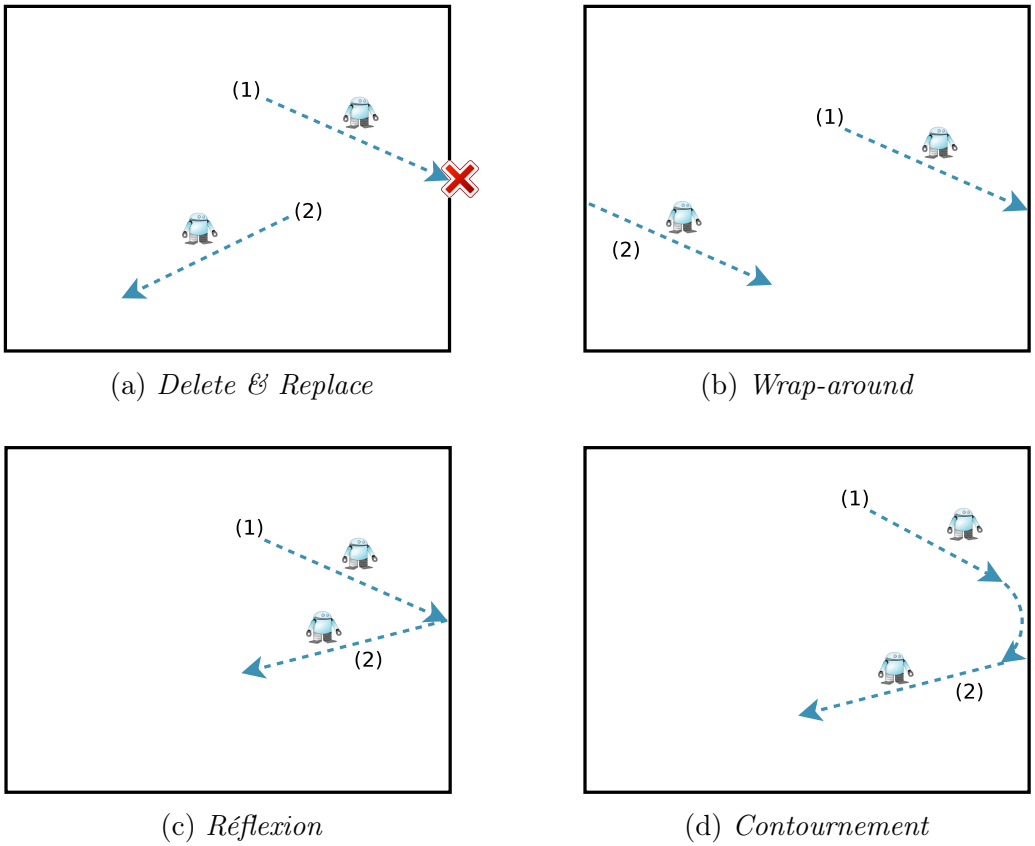
Avec le modèle *Random Walk*, les nœuds se déplacent dans une certaine direction pendant un temps donné, il se peut donc que les nœuds arrivent en limite de zone lors de ce déplacement. Les techniques permettant de réagir à cet événement sont les suivantes [163] :

— Le *delete & replace* consiste à supprimer le nœud quand il touche une bordure de la zone de déplacement ; un nouveau nœud apparaît alors pour remplacer le nœud perdu. La position initiale de ce nouveau nœud est choisie aléatoirement. Une illustration est fournie à la figure 3.3a. Cette technique est irréaliste car elle nécessite de remplacer un engin de la flotte à chaque contact avec une bordure de la zone.

— Le *wrap-around* consiste à faire réapparaître le nœud sur le coté opposé à la bordure touchée et dans la direction inverse. Une illustration est fournie à la figure 3.3b. Cette technique suppose que la zone est de forme torique.

— La *réflexion* consiste à faire rebondir l'engin sur la bordure et donc à lui faire faire une rotation. Cette technique est réaliste dans le sens où elle peut se matérialiser dans un scénario réel. Elle nécessite néanmoins que les engins aient la capacité d'effectuer une rotation sur place. Une illustration est donnée à la figure 3.3c.

— Le *contournement* consiste à modifier la direction de l'engin avant que ce dernier n'atteigne une bordure. Cette technique nécessite l'utilisation d'équipements particuliers (*e.g.* caméra, GPS) permettant au nœud de savoir qu'il se trouve à proximité d'une bordure. Cette technique est la plus réaliste car elle permet d'adapter le mécanisme d'évitement aux capacités de l'engin visé. Une illustration est fournie à la figure 3.3d.



(a) *Delete & Replace* (b) *Wrap-around*

(c) *Réflexion* (d) *Contournement*

FIGURE 3.3 – Illustration des différentes techniques d'évitement en bordures de zone.

La répartition des nœuds induite par le *Random Walk* est plus uniforme que celle induite par le *Random Waypoint* bien qu'il subsiste des parties "blanches" (*i.e.* où la densité est très faible) en bordure de zone [147, 150, 160].

En termes de réalisme, le modèle *Random Walk* est comparable au *Random Waypoint*. Les angles formés par les changements de direction peuvent être importants et donc ne pas être réalisables par certains types d'engins.

3.2.3 Random Direction

Le modèle *Random Direction* (RD) est un modèle dérivé du *Random Walk* où la durée de déplacement est infinie [142]. Par conséquent, les changements de direction sont effectués exclusivement aux bordures de la zone prédéfinie [147, 150]. Ce modèle repose donc sur une stratégie réflexive. Il existe quelques confusions dans la littérature entre le *Random Direction* et le *Random Walk* en raison de leur forte ressemblance [164, 165]. On peut considérer que le *Random Direction* est un cas particulier du *Random Walk* où le paramètre de durée de voyage n'existe pas. Le *Random Direction* fonctionne selon l'algorithme 3.3, une illustration du modèle est donnée en figure 3.4.

Les études faites sur ce modèle démontrent une meilleure uniformité dans la répartition des nœuds sur la zone que pour les modèles *Random Waypoint* et *Random Walk* mais au prix d'une connectivité plus faible [147, 160]. En revanche, le *Random*

Direction ne souffre pas de la présence de zones blanches, il dispose donc d'une meilleure couverture que les modèles précédents [147, 160].

Algorithm 3.3 Random Direction

Require: $maxSpeed, maxSleepTime, area$
  repeat
    $direction \leftarrow randomDirection()$
    $speed \leftarrow randomSpeed(maxSpeed)$
    $goToDirectionWithSpeedUntilAreaBorder(direction, speed, area)$
    $sleepTime \leftarrow randomTime(maxSleepTime)$
    $sleep(sleepTime)$
  end repeat

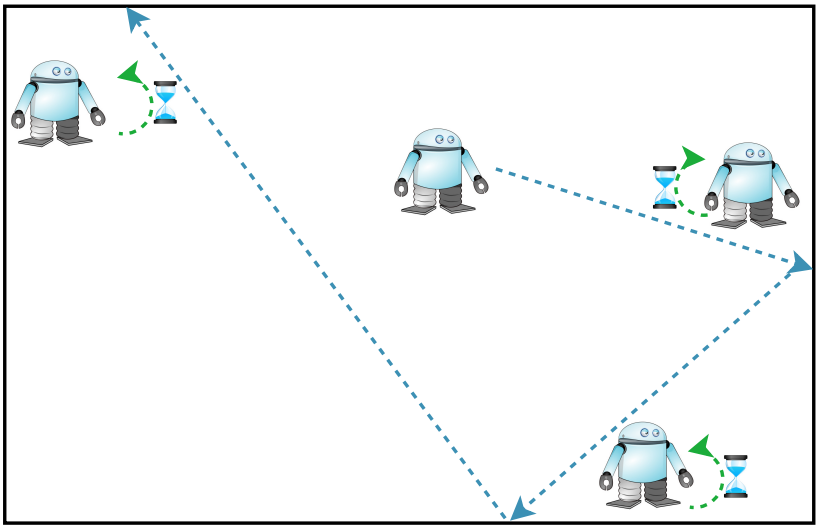


FIGURE 3.4 – Illustration du modèle de mobilité *Random Direction*.

Le modèle *Random Direction* peut, tout comme les modèles *Random Waypoint* et *Random Walk*, impliquer des virages serrés. Les conclusions sur le réalisme du *Random Waypoint* et du *Random Walk* sont donc également valables pour le *Random Direction*.

3.2.4 Gauss-Markov

Le modèle *Gauss-Markov* (GM) est similaire au modèle *Random Walk* mais impose des contraintes dans le choix des vitesses et des directions de déplacement [147, 150, 160]. En effet, ce modèle repose sur des fonctions prédéfinies qui déterminent la vitesse et la direction d'un déplacement en réutilisant les valeurs précédentes de ces paramètres. Le fonctionnement du modèle *Gauss-Markov* est décrit par l'algorithme 3.4 (les comportements en bordure de zone ne sont pas représentés dans cette figure). Les détails des fonctions de calcul de vitesse et de direction peuvent être trouvés dans l'étude de Tracy Camp *et al.* sur les modèles de mobilité les plus courants dans les réseaux de type MANets [147].

Algorithm 3.4 Gauss-Markov

Require: $maxSpeed, loopTime, area$
  repeat
    $direction \leftarrow nextDirection()$
    $speed \leftarrow nextSpeed(maxSpeed)$
    $goToDirectionWithSpeedForTime(direction, speed, loopTime, area)$
  end repeat

La gestion des bordures repose sur une stratégie réflexive ou bien sur une stratégie de contournement [150]. Ce modèle permet d'obtenir une répartition des nœuds et une connectivité comparables à celles du *Random Walk*.

La formule utilisée pour déterminer les directions permet de limiter les valeurs des angles lors des virages. De même, la formule de calcul des vitesses permet de limiter l'accélération des nœuds [147]. Ainsi, ce modèle permet de prendre en compte les spécificités des engins utilisés [150].

3.3 Modèles de mobilité spécifiques

Un modèle de mobilité dédié à un type d'engin particulier ou à un scénario précis est qualifié de *modèle de mobilité spécifique*. Dans cette section, nous présentons deux types de modèle spécifique :

— les *modèles urbains* qui sont dédiés aux véhicules terrestres évoluant dans un environnement contraint par des routes ;
— les *modèles aériens* qui sont spécifiques aux drones évoluant dans un environnement libre.

3.3.1 Modèles de mobilité urbains

Les modèles de mobilité dits *urbains* sont dédiés aux entités terrestres évoluant dans un environnement contraint (*i.e.* ayant des règles de déplacements, de sens de circulation, etc.) avec obstacles (*e.g.* des *buildings*). Nous présentons dans cette section deux exemples de ce type de modèle de mobilité spécifique.

Freeway

Freeway est un modèle de mobilité s'inspirant de la circulation autoroutière [148]. Dans ce modèle, les entités se déplacent en suivant des voies prédéfinies. Elles circulent dans un sens fixé (*i.e.* elles ne peuvent pas changer de sens) avec une vitesse variable mais en gardant une distance de sécurité avec les autres engins. Le modèle suppose deux voies de sens opposés contiguës. Par conséquent des communications peuvent être réalisées non seulement entre des véhicules à proximité sur la même voie mais également, lors des croisements, avec des engins présents dans la voie de sens opposée. Une illustration de ce modèle est proposée en figure 3.5.

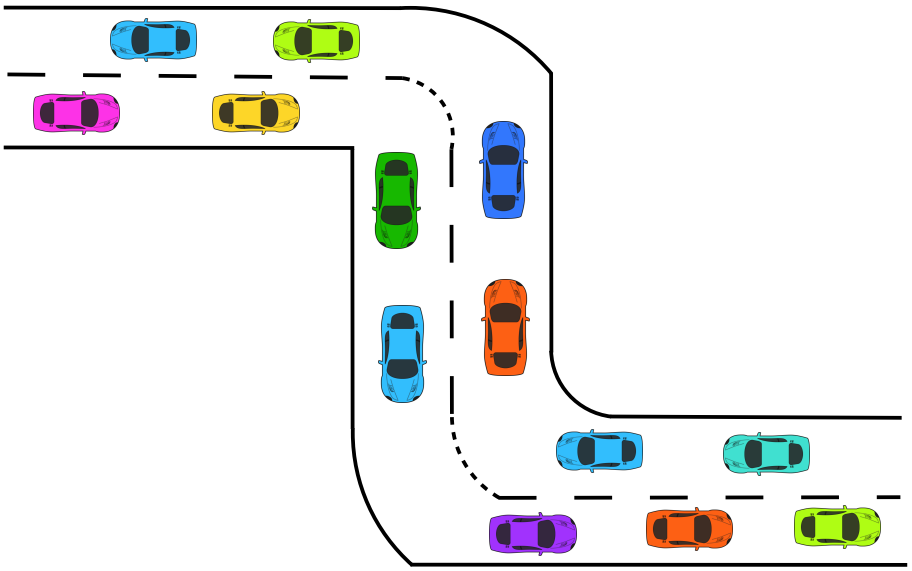


FIGURE 3.5 – Illustration du modèle de mobilité *Freeway*.

Ce modèle de mobilité implique une densité importante et une connectivité assez stable (*i.e.* les nœuds à proximité sur la même voie sont régulièrement connectés). Les variations de connectivité ont lieu lors des croisements de véhicules entre les deux voies et à cause des changements de vitesse des engins [148, 160].

Ce modèle est réaliste pour la modélisation d'un trafic autoroutier. Néanmoins, il suppose l'utilisation exclusive d'entités terrestres évoluant de manière plus ou moins coordonnée sur une route prédéfinie.

Manhattan

Le modèle *Manhattan* reproduit le comportement de véhicules terrestres évoluant dans une ville quadrillée de rues et constituant donc une grille [148]. Dans ce modèle, la carte utilisée est composée de voies à double sens (comme pour le modèle *Freeway*). La grille impose que les voies soient horizontales ou verticales. Le comportement des véhicules est le même que dans le modèle *Freeway* à la différence qu'ils disposent de la possibilité de changer de voie aux carrefours. Ce changement ne peut néanmoins pas être un demi-tour. Une illustration de ce modèle est proposée en figure 3.6.

Tout comme le modèle *Freeway*, le modèle *Manhattan* implique une densité importante, notamment aux carrefours. En revanche, les nœuds changent plus souvent de voisins car chacun de ces carrefours peut impliquer un changement des véhicules prédécesseurs et successeurs dans la voie empruntée [148, 160].

Ce modèle est réaliste dans le cadre d'une évolution urbaine. Tout comme le modèle *Freeway*, *Manhattan* suppose la coordination de véhicules terrestres évoluant sur des routes.

Conclusion

Les modèle *Manhattan* et *Freeway* sont des modèles qui contraignent fortement les véhicules dans leurs déplacements par l'utilisation de routes prédéfinies. Ces

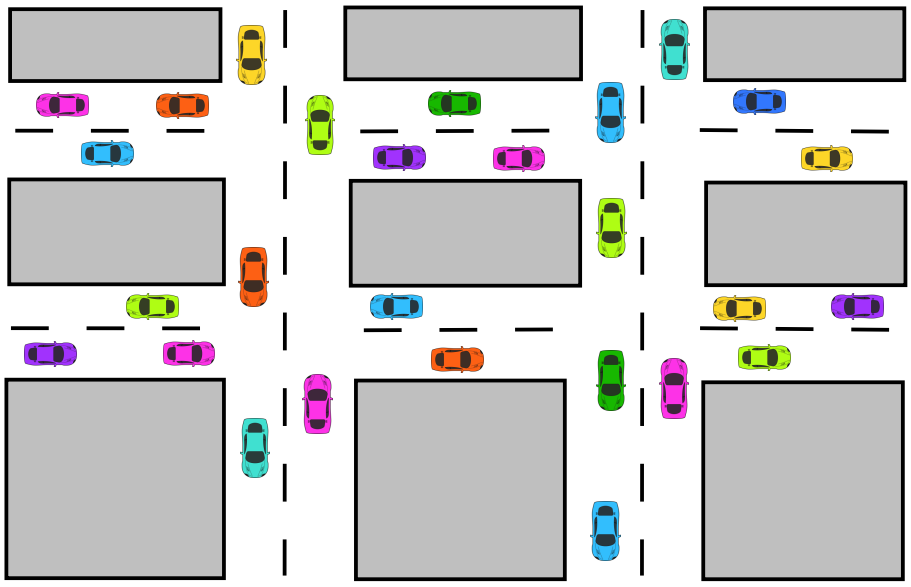


FIGURE 3.6 – Illustration du modèle de mobilité *Manhattan*.

contraintes permettent d'apporter une meilleure connectivité au réseau sous-jacent que celle induite par les modèles généralistes présentés dans la section 3.2 [148]. Ces modèles *Freeway* et *Manhattan* ne peuvent néanmoins être utilisés que dans des scénarios spécifiques aux milieux urbains.

3.3.2 Modèles de mobilité aériens

Les *modèles de mobilité aériens* sont dédiés aux entités de type drone évoluant dans un environnement libre. Ces modèles sont conçus pour respecter les contraintes aérodynamiques inhérentes aux engins aériens. En effet, comme nous l'avons évoqué en section 3.2.1, des virages trop serrés peuvent provoquer un décrochage de l'appareil. Par conséquent, il est nécessaire d'utiliser des modèles de mobilité qui prennent en compte cette particularité.

Semi-Random Circular

Le *Semi-Random Circular* (SRCM) est un modèle de mobilité développé pour des scénarios de patrouille et de recherche [166]. Un centre de rotation virtuel est défini dans la zone à couvrir ; ce centre est le même pour tous les engins durant toute la durée du scénario. Dans ce modèle, les engins suivent une trajectoire circulaire autour de ce centre de rotation. Une fois le cercle entièrement parcouru, un autre rayon est choisi et le processus est réitéré. Le fonctionnement du modèle SRCM est décrit par l'algorithme 3.5 et illustré à la figure 3.7.

Algorithm 3.5 Semi-Random Circular

Require: $maxSpeed, maxRadius, center$
  function SELECTCIRCLE( )
    $radius \leftarrow randomRadius(maxRadius)$
    $speed \leftarrow randomSpeed(maxSpeed)$
    $target \leftarrow ClosestTargetOnCircle(center, radius)$
    $goToTargetWithSpeed(center, radius, target, speed)$
  end function
  repeat
    SELECTCIRCLE( )
    while $(not\ circleCompleted())$ **do**
      $target \leftarrow nextTargetOnCircle(center, radius)$
      $speed \leftarrow randomSpeed(maxSpeed)$
      $goToTargetWithSpeedFollowingCircle(target, speed, center, radius)$
    end while
  end repeat

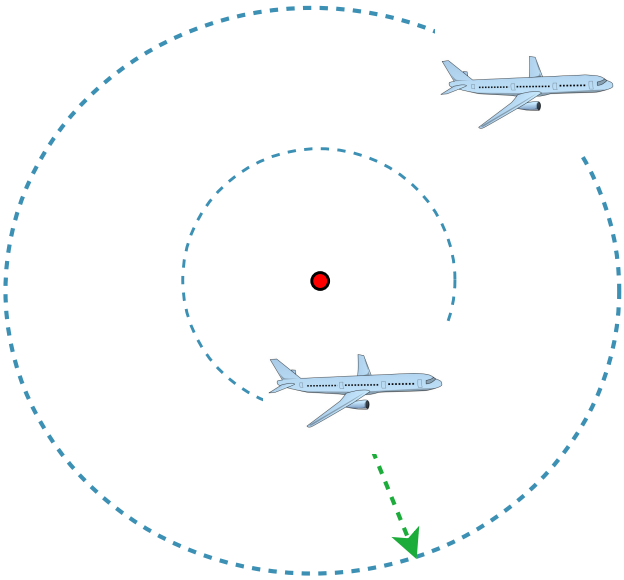


FIGURE 3.7 – Illustration du modèle de mobilité *Semi-Random Circular*.

Les analyses faites sur ce modèle montrent une uniformité dans la répartition des nœuds, dans la connectivité du réseau sous-jacent ainsi que dans la couverture de la zone considérée [149, 150, 166, 167]. En revanche, la contrainte d'un centre de rotation unique pour tous les engins limite son utilisation à des scénarios spécifiques (*e.g.* patrouille, recherche et sauvetage, surveillance) [165].

Ce modèle peut être considéré comme réaliste hormis lors des phases de changement de cercle de vol. En effet, les angles formés lors de cette étape peuvent être serrés (*i.e.* ce modèle, utilisé en conditions réelles, pourrait provoquer un décrochage de l'appareil lors de la phase de changement de cercle de vol). Cette partie du modèle devrait donc être modifiée pour lui assurer une plus grande conformité avec les contraintes aérodynamiques de certains engins aériens.

Smooth Turn

Le *Smooth Turn* (ST) est une évolution du modèle *Random Walk* qui contraint les angles lors des changements de direction [165, 167]. Ce modèle a été principalement développé pour modéliser les mouvements des engins aériens à voilure fixe. Il est néanmoins compatible avec les autres types de voilure. Le comportement du modèle *Smooth Turn* est décrit par l'algorithme 3.6 ; une illustration en est proposée en figure 3.8.

Algorithm 3.6 Smooth Turn

Require: $maxSpeed, maxTravelTime, maxRadius, minRadius, area$
  repeat
    $direction \leftarrow randomDirectionOnPerpendicularOfActualDirection()$
    $radius \leftarrow randomRadius(minRadius, maxRadius)$
    $center \leftarrow computePoint(direction, radius)$
    $speed \leftarrow randomSpeed(maxSpeed)$
    $time \leftarrow randomTime(maxTravelTime)$
    $FollowCircleWithSpeedForTime(center, radius, speed, time, area)$
  end repeat

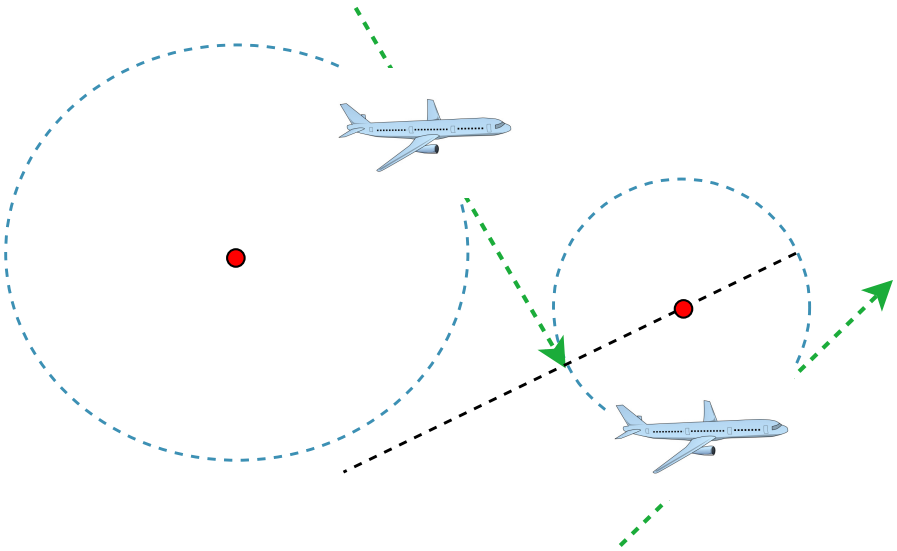


FIGURE 3.8 – Illustration du modèle de mobilité *Smooth Turn*.

Les études et analyses effectuées sur ce modèle montrent une uniformité dans la répartition des nœuds ainsi que dans la connectivité du réseau sous-jacent [149, 150, 165, 167]. La technique utilisée lorsqu'un engin rencontre une bordure de la zone considérée (e.g. *wrap-around*, *réflexion*) ne semble pas avoir d'impact sur ces résultats [165, 167].

Ce modèle est réaliste dans la mesure où ses paramètres permettent d'exprimer les angles minimaux et maximaux autorisés pour les engins. Une version de ce modèle existe en trois dimensions permettant ainsi d'inclure la mobilité des nœuds lors des phases de changement d'altitude telles que le décollage ou l'atterrissage [168].

Conclusion

De par la prise en compte des contraintes aérodynamiques des engins aériens, SRCM et ST sont des modèles de mobilité réalistes pour ce type d'appareils. Ils sont donc à privilégier pour des expérimentations impliquant des drones. De plus, ces modèles possèdent des propriétés intéressantes en ce qui concerne la répartition des nœuds ainsi que la connectivité du réseau sous-jacent.

3.4 Modèles de mobilité hybrides

Les *modèles de mobilité hybrides* regroupent deux sous-genres distincts.

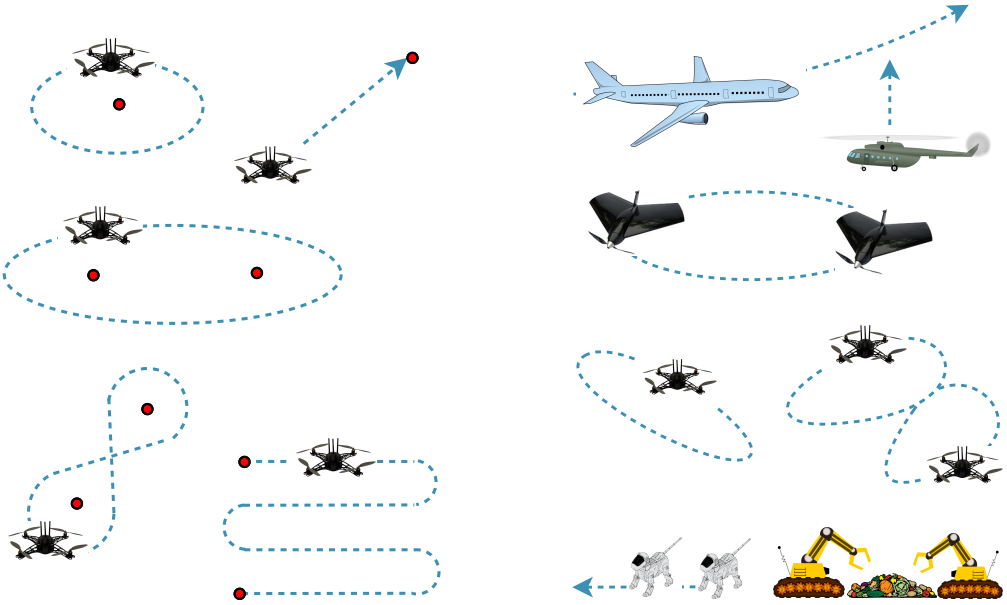
Un modèle de mobilité est dit *complexe* lorsqu'il contient plusieurs séquences de mouvements différents. Par exemple, le modèle de mobilité intégré à l'auto-pilote *Paparazzi* (PPRZM) développé par l'ENAC décrit 5 types indépendants de mouvements qui peuvent être effectués par un drone à voilure tournante [169] :

- *stationnaire* : le drone effectue des rotations autour d'un point d'intérêt avec un rayon fixé ;
- *cible* : le drone se dirige vers une cible en suivant une trajectoire rectiligne ;
- *huit* : le drone suit une trajectoire représentant un "8" dont les centres et le rayon des boucles sont fixés ;
- *scanne* : le drone parcourt horizontalement une zone rectangulaire en inversant sa direction lors de l'approche de la bordure de la zone ;
- *ovale* : le drone suit une trajectoire ovale dont le centre et les dimensions sont fixés.

Dans le modèle PPRZM, chaque type de mouvement est associé à une probabilité. Cette dernière est utilisée pour choisir la prochaine séquence à effectuer. Une illustration du modèle *Paparazzi* est fournie en figure 3.9a.

Lorsqu'un scénario implique l'utilisation d'engins obéissant à des modèles de mobilité différents, on parle de modèle *multiple*. Certains scénarios reposent par exemple sur l'utilisation de plusieurs engins aériens dont les trajectoires sont calculées et effectuées différemment [170]. D'autres scénarios, principalement militaires, reposent sur des flottes hybrides (*i.e.* engins terrestres et aériens) dont les entités ne suivent pas le même modèle de mobilité [171, 172]. Une illustration d'un tel scénario est proposée en figure 3.9b.

Les modèles de mobilité hybrides, de par leur complexité et leurs possibilités de combinaisons, sont difficiles à analyser [149, 150]. Une caractérisation commune (*e.g.* connectivité, couverture, répartition) est donc impossible à obtenir.



(a) *Complexe* (b) *Multiple*

FIGURE 3.9 – Illustration des modèles de mobilité *hybrides*.

3.5 Résumé

Dans ce chapitre nous avons présenté un ensemble de *modèles de mobilité* de natures différentes. Nous avons décrit les *modèles de mobilité généralistes* les plus courants dans la littérature pour les réseaux MANets. Ainsi, nous avons présenté les modèles *Random Waypoint*, *Random Walk*, *Random Direction* et *Gauss-Markov*. Nous avons détaillé leur fonctionnement et avons discuté de leurs propriétés (*i.e.* connectivité, distribution, couverture) et de leur réalisme. Nous avons mis en évidence que ces modèles n'étaient pas adaptés à tous les types d'engin notamment lors des changements de direction. Nous avons également décrit certains *modèles de mobilité spécifiques* aux engins terrestres en *milieu urbain*. Enfin, nous avons présenté des *modèles dédiés aux engins aériens* de type drone. Nous avons constaté que ces modèles de mobilité spécifiques permettaient de mieux prendre en compte les particularités des engins visés. En revanche, ils restreignent les mouvements des engins modélisés et ne sont donc pas satisfaisant pour tous les scénarios. Nous avons enfin défini les *modèles de mobilité hybrides* qui combinent plusieurs modèles de mobilité pour une seule entité (*i.e. modèle complexe*) ou pour une flotte hétérogène (*i.e. modèle multiple*).

Tracy Camp *et al.* ont montré que le modèle de mobilité utilisé dans un scénario d'expérimentation a un impact significatif sur les résultats obtenus [147]. Par conséquent, nous utiliserons un modèle de mobilité proche de la réalité afin d'obtenir des résultats cohérents lors de nos phases d'évaluation. De plus, nous effectuerons nos mesures avec plusieurs modèles de mobilité dans le but de renforcer la validité de nos résultats.

CHAPITRE 4

Découverte de services

Sommaire

4.1 Introduction

Nous avons défini dans le chapitre 1 la notion de *capacité* pour les systèmes autonomes (*c.f.* définition 1.5). Pour rappel, une capacité représente l'ensemble des équipements qui permettent à une entité de réaliser une tâche prédéfinie de manière autonome. Dans la cadre d'une flotte hétérogène, les capacités hébergées par les entités peuvent différer d'un engin à l'autre. Pour effectuer certaines tâches complexes, il est donc nécessaire de fournir aux engins de la flotte un mécanisme leur permettant de faire appel aux capacités des autres. Une capacité ainsi partagée est appelée un *service*. Nous définissons un service comme ci-dessous :

Définition 4.1. *Un* ***service*** *est une capacité particulière proposée par une entité à l'ensemble (ou à un sous ensemble) des membres de la flotte. Ce service peut être utilisé par une entité autre que son porteur afin de réaliser une tâche spécifique.*

Le mécanisme permettant le partage de ces services est appelé *découverte de services*. Trivialement, la découverte de services au sein d'un réseau consiste pour une entité x (*i.e.* le client) à localiser un service s se trouvant sur une entité y (*i.e.* le fournisseur). La découverte de service ne peut néanmoins se cantonner à la seule

fonction de localisation. Ververidis et Polyzos définissent la découverte de services comme l'ensemble des fonctionnalités suivantes mises en œuvre par chaque nœud du réseau [173] :

— l'annonce de ses propres services aux autres membres du réseau ;
— la localisation des services proposés dans le réseau ;
— la sélection du fournisseur le plus pertinent dans le cas où un même service serait proposé par plusieurs entités ;
— l'invocation du service ainsi découvert et sélectionné.

Néanmoins, la majorité des mécanismes de découverte de services existants ne fournissent que la fonction de localisation, ne traitant pas les fonctions de sélection et d'usage des services découverts [174, 175].

Dans le cadre des réseaux MANets, la principale difficulté relative à la mise en place d'un mécanisme de découverte de services réside dans la mobilité des nœuds [173]. Comme nous l'avons constaté aux chapitres 2 et 3, cette mobilité implique des changements topologiques importants et récurrents. Couplée à la volatilité des nœuds, la mobilité peut avoir des effets néfastes sur la qualité des communications au sein du réseau. Or, les mécanismes de découverte de services nécessitent des communications entre clients et fournisseurs de services ; ils sont donc également tributaires de la mobilité et de la volatilité des nœuds.

4.2 Architectures pour la découverte de services

La littérature compte trois types d'architectures pour les mécanismes de découverte de services [173, 174, 176] : basé sur un annuaire (*directory-based*), sans annuaire (*directory-less*) et hybride.

4.2.1 Modèles avec annuaire

Un *annuaire* regroupe l'ensemble des informations relatives à la localisation des services fournis par les nœuds du réseau. En d'autres termes, c'est une base de données associant les services disponibles aux fournisseurs les hébergeant. Dans un modèle basé sur un annuaire, un client x voulant invoquer un service s s'adresse à l'annuaire pour obtenir les informations relatives au(x) fournisseur(s) du service s. Cet annuaire peut être stocké de façon *centralisée* ou bien *distribuée*.

Modèle centralisé

Le modèle avec annuaire est dit *centralisé* lorsque la totalité de l'annuaire est hébergée par une entité unique, facilitant ainsi l'enregistrement et la consultation des services proposés dans le réseau. L'enregistrement est le mécanisme permettant à un fournisseur d'indiquer à l'annuaire l'ensemble des services qu'il propose [174]. Le modèle centralisé a historiquement été développé pour les réseaux filaires [173]. Ainsi, il ne prend pas en compte certaines particularités des réseaux MANets comme

la mobilité et la volatilité de nœuds [175]. En effet, cette mobilité peut induire une impossibilité de communication entre un client et l'annuaire empêchant ainsi les services d'être découverts. Par ailleurs, une panne de l'entité hébergeant l'annuaire rendrait le système inopérant. Le standard de découverte de services *UDDI* [177] repose, par exemple, sur l'utilisation d'un annuaire centralisé.

Modèle distribué

Dans le modèle avec annuaire *distribué*, plusieurs (ou l'ensemble des) nœuds du réseau hébergent une partie de l'annuaire. Plusieurs approches du modèle distribué existent [173, 174]. Dans cette section, nous présentons les plus courantes.

L'*approche multi-annuaires* repose sur l'utilisation de plusieurs annuaires, chacun hébergé par un nœud différent. Cette approche permet de réduire l'impact des pannes éventuelles qui peuvent survenir sur les nœuds porteurs d'une partie de l'annuaire. Ce modèle n'implique pas de communication entre les porteurs d'annuaires et par conséquent le contenu de chaque annuaire peut être différent de celui des autres, les fournisseurs étant libres du choix des annuaires sur lesquels ils enregistrent leurs services respectifs. De plus, la communication entre un annuaire et un fournisseur n'est pas toujours possible. Cette solution n'est donc pas adaptée pour les réseaux fortement dynamiques tels que les MANets. Le système *JINI/River* est un exemple de protocole de découverte de services reposant sur une approche de ce type [178].

L'*approche par réplication* est une extension de l'approche multi-annuaires qui intègre des mécanismes de synchronisation entre ces derniers. Cette synchronisation permet de répliquer les informations contenues dans les différents annuaires afin de maximiser les chances pour un client de trouver un service recherché par l'intermédiaire de n'importe quel annuaire présent dans le réseau. Ces mécanismes de synchronisation impliquent néanmoins des communications régulières entre les annuaires ainsi qu'un trafic réseau élevé en raison de la quantité d'informations échangées entre ces derniers. *SDP* [179] est un exemple bien connu de mécanisme de découverte de services basé sur cette approche par réplication d'annuaire.

L'*approche par ensemble dominant (backbone)* repose également sur l'utilisation de plusieurs annuaires. Les nœuds hébergeant ces annuaires sont élus de façon distribuée parmi l'ensemble des entités du réseau de manière à former un sous-réseau connexe (le *backbone*). Contrairement à l'approche par réplication, l'approche par ensemble dominant ne requiert pas la synchronisation du contenu des annuaires hébergés par les nœuds du *backbone*. En revanche, ces nœuds communiquent entre eux afin de relayer les requêtes qu'ils ne peuvent pas résoudre. Ainsi, l'enregistrement ou la consultation d'un service n'ont besoin d'être effectués qu'auprès d'un unique annuaire, ce dernier pouvant transmettre les requêtes aux autres membres du *backbone* si les services recherchés ne sont pas hébergés par lui même. La formation et le maintien du *backbone* requiert l'utilisation de messages de contrôle. Une forte mobilité ainsi qu'une volatilité importante des nœuds du *backbone* peuvent donc augmenter le volume du trafic de contrôle. Le choix de l'annuaire destinataire d'une requête (pour enregistrement ou consultation) peut être fait de plusieurs façons : le protocole *DSDP* [180] utilise une fonction aléatoire tandis que le protocole *SSD* [181] réutilise des informations gardées en cache pour sélectionner l'annuaire le

plus pertinent.

L'*approche hiérarchique (par cluster)* repose sur la segmentation du réseau en sous-groupes possédant chacun son annuaire propre. Tout comme l'approche par ensemble dominant, les annuaires communiquent entre eux et peuvent ainsi relayer les requêtes aux autres annuaires. Ici, chacun des groupes possède son annuaire propre et ne peut communiquer qu'avec lui. La formation et le maintien des groupes requièrent également l'échange régulier de messages de contrôle entre les nœuds du réseau. Une forte mobilité ainsi qu'une volatilité importante impliquent une restructuration récurrente des groupes qui peuvent paralyser le système de découverte de services. Les groupes peuvent être constitués selon des critères variés comme le type de services [182], la proximité géographique des nœuds [183] où encore le type de mobilité des nœuds [184].

L'*approche par DHT (Distributed Hash Table)* utilise la totalité des nœuds du réseau pour stocker les différentes parties de l'annuaire [185]. Celui-ci est ainsi subdivisé et réparti sur l'ensemble des nœuds du réseau. Une fonction de distance, basée sur un identifiant unique dont est doté chaque nœud, permet de déterminer sur lequel d'entre eux une certaine partie de l'annuaire doit être stockée. Cette fonction est également utilisée lorsqu'un client souhaite connaître le(s) fournisseur(s) d'un service. Elle permet dans ce cas de retrouver les nœuds porteurs de la partie concernée de l'annuaire. La fonction de distance permet de calculer une distance logique (le nombre de sauts ou la distance géographique n'ont aucun impact sur cette fonction) entre deux identifiants. Les DHT reposent sur un réseau virtuel (*overlay*) structuré à l'aide de la fonction de distance utilisée. Une mobilité ou une volatilité importante des nœuds peuvent impliquer des restructurations dans l'*overlay* engendrant un trafic réseau soutenu ainsi qu'une redistribution des parties de l'annuaire entre les nœuds. La réalisation de cette phase entraîne une chute importante du taux de succès des requêtes effectuées dans la DHT [186]. Cette approche n'a pas été conçue pour les réseaux de type MANet mais pour les réseaux statiques à grande échelle (avec tolérance aux pannes). On trouve de nombreuses solutions basées sur des DHT dans la littérature [187]. Elles différent notamment par la structure de l'*overlay* ainsi que par la fonction de distance utilisée. *Kademlia* [188] utilise par exemple l'opérateur *ou exclusif* (*XOR*) comme fonction de distance. *Chord* [189] repose quant à lui sur un *overlay* en anneau. Enfin, *Cloak* [190] utilise la géométrie hyperbolique pour structurer son *overlay*.

Les approches par annuaire distribué reposent généralement sur l'utilisation de routes calculées par des protocoles de routage opérant sur le réseau sous-jacent afin d'établir les communications entre clients, annuaires et fournisseurs de services. Les performances des protocoles de routage utilisés ont donc un impact sur celles des mécanismes de découverte de services mis en œuvre dans le réseau. Les différentes analyses montrent que le modèle distribué est adapté aux réseaux de grande taille ayant une faible mobilité [173, 174].

Les changements topologiques impliquent des modifications dans la structure du mécanisme de découverte (*backbone*, *cluster*, *overlay*). Lors de ces restructurations, une redistribution des informations contenues dans les annuaires est opérée. Cette redistribution peut être plus ou moins complexe à mettre en œuvre et donc prendre plus ou moins de temps. Durant cette phase, le temps de réponse ainsi que le taux

d'échec des requêtes de services augmentent. Par conséquent, une trop grande mobilité provoque un état de restructuration permanent et rend donc ces systèmes instables. À notre connaissance, aucune mesure expérimentale sur l'impact de la volatilité des nœuds n'a été faite sur l'ensemble de ces approches. Nous pensons néanmoins qu'une volatilité accrue entraînerait une instabilité des approches distribuées au même titre qu'une mobilité trop importante.

4.2.2 Modèle sans annuaire

Dans un modèle sans annuaire, chaque fournisseur est responsable de l'annonce de ses propres services au reste des membres du réseau. Les communications entre clients et fournisseurs peuvent reposer sur des routes établies par des protocoles de routage. Elles peuvent aussi reposer exclusivement sur une stratégie d'inondation [173, 175]. Le modèle sans annuaire est plus tolérant à la mobilité que les approches présentées en section 4.2.1. En revanche, il n'est pas conçu pour être utilisé sur des réseaux de taille importante [173, 174]. Il existe plusieurs modes de fonctionnement pour ce modèle : *push*, *pull* [191] et *hybride* [192].

Mode *push*

Dans le mode *push*, les fournisseurs diffusent la liste des services qu'ils proposent par inondation. De ce fait, un client est en mesure de connaître (avec une forte probabilité) le fournisseur d'un service donné. Le mode *push* est une stratégie proactive dans laquelle chaque client héberge un annuaire local et répertorie l'ensemble des services disponibles qui lui ont été communiqués au travers des annonces des fournisseurs. *DEAPspace* [193] est un exemple de protocole de découverte de services sans annuaire fonctionnant exclusivement en mode *push* par inondation. *OLSR-SD* [194] est quant à lui une extension du protocole de routage proactif OLSR intégrant à la fois les fonctions de routage et de découverte de services en mode *push*.

Le volume du trafic réseau induit par le mode *push* dépend du nombre de fournisseurs et de services présents dans le réseau [192, 195]. En effet, chaque fournisseur diffusant par inondation et à intervalles de temps réguliers la liste de ses services, une augmentation du nombre de fournisseurs entraîne une augmentation du trafic réseau général. Cela peut donc engendrer une saturation des communications. Les stratégies d'inondation contrôlée présentées dans la section 2.3.2 permettent néanmoins de réduire l'impact de ce phénomène.

Mode *pull*

Dans le mode *pull*, les clients n'ont pas besoin de connaître l'identité des fournisseurs à l'avance. Lorsqu'un client x souhaite invoquer un service s se trouvant sur un fournisseur y, le client x inonde le réseau avec une requête de localisation du service s. Le message de requête est dupliqué et retransmis par les autres nœuds jusqu'à atteindre le fournisseur y. Ce dernier répond avec un message *unicast* si un protocole de routage est utilisé sur le réseau sous-jacent et par inondation dans le cas contraire. Les protocoles de découverte de services exclusivement en mode *pull*

sont généralement des extensions de protocoles de routage réactifs. *AODV-SD* et *DSR-SD* en sont des exemples [196].

Le trafic réseau induit par le mode *pull* dépend du nombre de clients ainsi que du nombre de requêtes de découverte émises [192, 195]. Une augmentation de ces valeurs intensifie en effet considérablement ce trafic en raison de la duplication des requêtes émises par inondation. Les stratégies présentées dans la section 2.3.2 permettent, tout comme pour le mode *push*, de réduire l'impact de ce phénomène.

Mode hybride

Le mode *hybride* utilise à la fois le mode *push* et le mode *pull* en fonction des messages. Le mode hybride a par exemple été implémenté en tant qu'extension de protocoles de routage existants tels que *M-ZRP* [197, 198] qui repose sur le protocole de routage hybride ZRP. Mais d'autres systèmes de découverte de services hybrides comme *Konark* [199, 200] et *Allia* [201] fonctionnent indépendamment du protocole de routage utilisé sur le réseau sous-jacent.

Certains modèles hybrides dits *adaptables* permettent d'ajuster dynamiquement le mode (*i.e. push* ou *pull*) en fonction de divers paramètres (*e.g.* nombre de clients, nombre de fournisseurs, nombre de services, nombre de requêtes). Ce type de modèle vise à réduire l'impact des problèmes de surcharge réseau évoqués pour les modes *push* et *pull* en selectionnant dynamiquement le mode le plus adapté au contexte du réseau [192].

4.2.3 Modèle hybride

Un modèle hybride (à ne pas confondre avec le mode hybride du modèle sans annuaire) fonctionne avec ou sans annuaire. Dans ce modèle, un client souhaitant invoquer un service s s'adresse en priorité à un annuaire pour effectuer sa requête de découverte. Si aucun annuaire n'est disponible ou si le service s n'est pas trouvé, le client utilise une stratégie sans annuaire pour obtenir la localisation du service s. Peu de systèmes hybrides existent dans la littérature. Nous pouvons néanmoins citer *LSD* [202] (*Lightweight Service Discovery*) qui est une extension du protocole de routage proactif OLSR. Bien qu'il ait été conçu pour des réseaux filaires, le standard *SLP* (*Service Location Protocol*) est également un système de découverte de services hybrides.

4.2.4 Discussion

Ververidis et Polyzos affirment qu'aucun consensus sur ce que serait la meilleure architecture n'a pour le moment été atteint [173]. Néanmoins certaines analyses tendent à montrer que la mobilité des nœuds ainsi que la taille du réseau sont les paramètres les plus cruciaux dans le choix d'une architecture pour la découverte de services. Ainsi, Mian et *al.* estiment que les architectures avec annuaire basées sur un modèle distribué permettent un passage à des échelles importantes tandis que les architectures sans annuaire basées sur une stratégie d'inondation sont plus adaptées à des réseaux à forte mobilité [174]. La figure 4.1 résume les résultats de ces analyses.

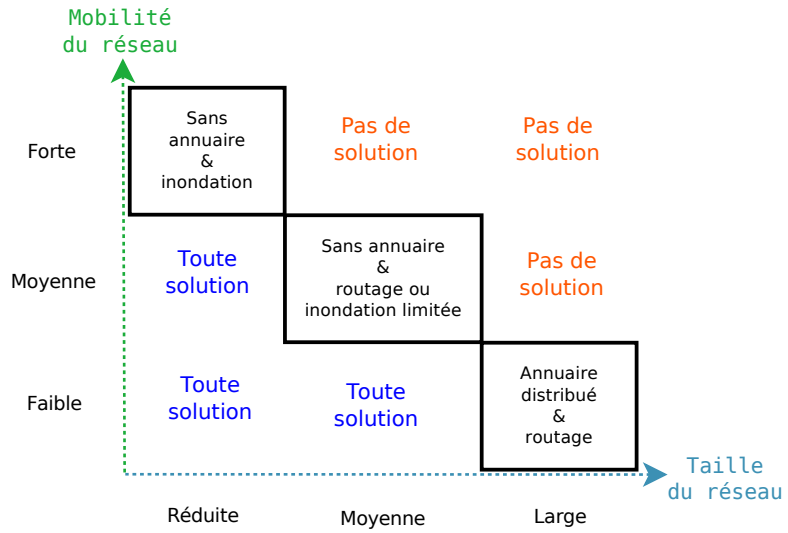


FIGURE 4.1 – Choix de l'architecture d'une solution de découverte de services en fonction de la mobilité de de la taille du réseau.

Nous pouvons constater qu'il n'existe pas de solution efficace pour un réseau de taille importante et fortement mobile. Comme nous l'avons évoqué dans la section 4.2.1, une forte volatilité des nœuds pourraient engendrer des effets comparables à ceux causés par une forte mobilité pour les architectures basées sur un annuaire. Nous pouvons donc considérer que la dynamique (*i.e.* la mobilité et la volatilité des nœuds) ainsi que la taille du réseau sont les paramètres à considérer dans le choix d'une architecture pour la découverte de services. La figure 4.2 prend en compte cette considération et simplifie la figure 4.1 en ne se focalisant que sur les solutions optimales.

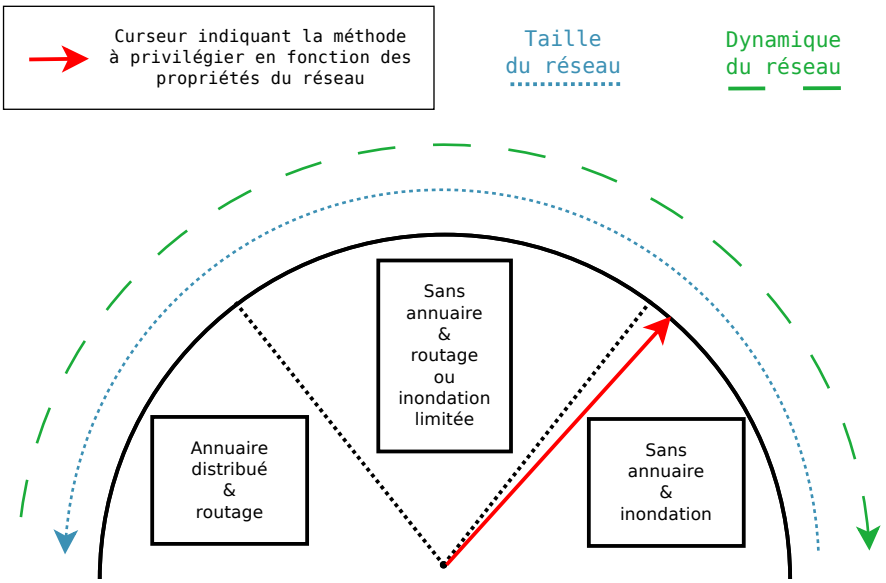


FIGURE 4.2 – Diagramme décisionnel pour le choix de l'architecture d'une solution de découverte de services.

4.3 Résumé

Dans ce chapitre, nous avons défini la notion de *découverte de services* dans un réseau mobile et avons présenté les différentes *architectures* existantes répondant à cette problématique. Nous avons détaillé les *modèles* et les *approches* développés dans la littérature pour chacune de ces architectures. Nous avons mis en évidence que le contexte (*e.g.* mobilité des nœuds, taille du réseau) a un impact significatif sur les performances de ces systèmes.

Dans cette étude, nous nous intéressons aux flottes fortement dynamiques (*i.e.* volatiles et mobiles) et de taille raisonnable. Par conséquent, nous nous concentrerons sur le modèle sans annuaire reposant exclusivement sur une stratégie d'inondation.

II Approche proposée : AMiRALE

CHAPITRE 5

Mise en perspective des travaux de cette thèse

Sommaire

Dans ce chapitre, nous précisons l'intérêt et la nature des travaux menés dans cette thèse, sous l'éclairage des mécanismes présentés précédemment.

5.1 Des services aux tâches

La notion de service n'a pas la même signification dans un réseau MANet généraliste que dans une flotte autonome. En effet, l'invocation d'un service par un client dans un réseau classique vise à obtenir une information. Par exemple l'utilisation du service DNS [203, 204] a pour but d'obtenir l'adresse IP associée à un nom de domaine. Dans une flotte autonome, l'invocation d'un service telle que nous l'avons définie en section 4.1 permet de déléguer une tâche à une entité ayant les capacités nécessaires pour la résoudre.

Le principe de collaboration entre entités autonomes que nous proposons se base sur les notions classiques de *rôle* [205] et de *tâche* [206]. Dans le contexte des flottes autonomes, nous précisons ces deux termes de la manière suivante :

Définition 5.1. *Une **tâche** est un objectif qui doit être réalisé par une ou plusieurs entités de la flotte.*

Prenons par exemple un groupe de robots devant inspecter l'intérieur d'un bâtiment. Une tâche pourrait être l'inspection d'une pièce particulière.

Définition 5.2. *Un **rôle** est l'ensemble des tâches qui peuvent être attribuées à une entité de la flotte en tenant compte de ses capacités propres.*

Une fois les notions ci-dessus ainsi définies, nous pouvons les utiliser pour préciser dans ce contexte l'objectif de notre étude. Il s'agit de proposer un mécanisme de découverte de services permettant aux engins appartenant à une flotte fortement dynamique de collaborer. Les fonctionnalités de sélection et d'invocation d'un service introduites dans la section 4.1 sont transposables à un problème d'*allocation de tâches*. En effet, l'invocation d'un service a ici pour objectif d'allouer une tâche à son fournisseur, l'entité appelante ne disposant pas des capacités nécessaires à sa résolution.

L'allocation de tâches est un problème $\mathcal{NP}$-difficile [207, 208] pour lequel un grand nombre d'heuristiques ont été développées telles que l'*approche par priorité* [209], l'*approche probabiliste* [210] ou encore l'*approche par négociation* [211] (*i.e. market-based*). Néanmoins, ces approches ne sont pas adaptées aux réseaux fortement dynamiques auxquelles nous nous intéressons. En effet, elles sont conçues pour fonctionner de façon centralisée ou dans un réseau faiblement dynamique et où la communication est possible entre tous les engins de la flotte [68, 212].

Les approches par *sélection de tâches* permettent aux entités de choisir elles mêmes leurs tâches. Pour cela, les entités se communiquent plus ou moins régulièrement entre elles l'ensemble des tâches à effectuer par la flotte. Plusieurs algorithmes proposent des solution au problème de la sélection de tâches. Ils différent principalement par le critère qui permet d'effectuer cette sélection : l'ancienneté des tâches [213], la priorité des tâches [214] ou encore l'utilisation d'un modèle probabiliste [215]. Le mécanisme collaboratif que nous proposons sera basé sur le principe de sélection de tâches et permettra de spécifier le critère de l'algorithme le plus pertinent pour le scénario visé.

Contrairement à un service, une tâche peut prendre plusieurs formes [216]. Par exemple, une tâche *élémentaire* est atomique, elle ne peut donc pas être subdivisée. Comme nous l'avons défini en section 4.1, un service représente une capacité particulière proposée par une entité aux autres membres de la flotte. Nous considérons ici que l'invocation d'un service revient à demander à son fournisseur d'effectuer une tâche élémentaire. Nous ne considérerons donc pas les autres formes de tâches dans cette étude.

La façon dont les tâches peuvent (ou doivent) être réalisées peut également prendre plusieurs formes [207, 206] :

— une tâche peut nécessiter l'action d'*une seule* (*i.e.* un modèle d'interaction en collaboration) ou bien de *plusieurs* (*i.e.* un modèle d'interaction en coalition) entités ;
— une entité peut être *mono-tâche* ou bien *multi-tâches* ;
— il peut exister une relation de *dépendance entre les tâches*.

Nous nous intéressons à un modèle collaboratif : nous considérons que chaque tâche ne nécessite l'action que d'une seule entité pour être résolue.

Enfin, une allocation de tâches effectuée sur un ensemble d'entités peut nécessiter une redistribution des tâches en cas de panne d'une ou de plusieurs entités [55]. Notre mécanisme devra donc intégrer une fonction ré-allocation afin d'assurer la tolérance aux pannes qui pourraient survenir sur les entités de la flotte.

Comme nous l'avons vu dans la section 4.2, le choix d'une approche particulière pour la découverte de services dépend des caractéristiques du réseau sous-jacent (*e.g.* mobilité des nœuds, taille du réseau). Nous avons vu dans la section 2.3 que cette observation était également valable dans le choix de la stratégie de communication employée entre les nœuds du réseau.

Or, Polyzos *et al.* ont constaté qu'aucun mécanisme de découverte de services ne prenant en compte les caractéristiques du réseau sous-jacent n'avait été développé [217]. Ils ont ainsi mis en évidence trois éléments internes à ces mécanismes qui devraient, selon eux, être adaptables dans un réseau d'entités autonomes :

— la stratégie de *communication* des services, *i.e.* la manière dont les informations sur les services sont diffusées par les nœuds ;
— la stratégie de *sélection* des services, *i.e.* la manière dont un service est choisi en cas de propositions multiples ;
— la stratégie de *récupération* des services, *i.e.* la manière dont les fonctions offertes par un service sont rétablies ou récupérées en cas de panne.

C'est pourquoi nous proposons dans cette thèse un nouveau mécanisme de découverte de services adapté aux réseaux d'entités autonomes fortement dynamiques.

5.2 Problématique visée

Dans cette thèse, nous nous intéressons aux flottes strictement autonomes d'engins mobiles communicants. Plus spécifiquement, nous nous concentrons sur la problématique de la collaboration entres les entités qui composent une telle flotte. Le mécanisme permettant cette collaboration doit nécessairement être distribué en raison de la nature entièrement autonome du système.

Nous avons montré dans ce chapitre que les mécanismes de découverte de services existants n'avaient pas été conçus pour un tel paradigme. Nous avons néanmoins proposé une expression du principe de découverte de services, basée sur les notions de tâche et de rôle, permettant ainsi de transposer ce concept aux flottes autonomes collaboratives.

Nous avons également montré au travers des chapitres 1 à 5 que le problème de la collaboration dans une flotte autonome soulève un ensemble de sous-problèmes distincts tels que la communication inter-engins, la découverte des services ou encore l'allocation de tâches. Nous avons mis en évidence qu'il n'existe pas de solution à toutes ces problématiques qui soit satisfaisante quel que soit le contexte d'utilisation en termes des caractéristiques de la flotte concernée. En effet, nous avons vu au travers des chapitres 2 et 3 que la mobilité des nœuds a un impact significatif sur la connectivité du réseau. Nous avons enfin constaté que l'environnement (*i.e.* la présence d'obstacles, la distance entre les nœuds, les pannes imprévisibles) ainsi que la technologie de communication utilisée entre les entités de la flotte (*i.e.* la portée radio, les collisions, la congestion et la contention réseau) pouvaient amplifier ce phénomène. Les communications peuvent être ainsi sporadiques et la volatilité des nœuds accrue.

Fort de l'ensemble de ces constations, nous souhaitons proposer un nouveau mécanisme collaboratif pour les flottes fortement dynamiques. Notre proposition devra être adaptable aux caractéristiques du réseau sous-jacent ainsi qu'au scénario visé comme nous l'avons expliqué dans la section 5.1. De plus, les flottes auxquelles nous nous intéressons peuvent disposer d'un niveau d'hétérogénéité variable (*e.g.* différents types d'engins, des capacités différentes d'un engin à l'autre). En revanche, nous nous plaçons dans le cadre d'une flotte de taille raisonnable (*i.e.* inférieure à 100 engins). Cette restriction peut être considérée comme réaliste et acceptable dans la mesure où la majorité des flottes de systèmes autonomes existantes n'excèdent pas la dizaine d'appareils.

5.3 Approche proposée

Nous proposons donc dans cette thèse un nouveau mécanisme collaboratif inspiré des principes de la découverte de services. Ce mécanisme permettra la collaboration entre les engins d'une flotte strictement autonome de taille raisonnable. Notre mécanisme sera résilient à des conditions extrêmes telles qu'une forte volatilité des nœuds et une forte mobilité de la flotte qui engendrent une faible connectivité du réseau sous-jacent et donc des communications sporadiques. En raison de ces contraintes, nous utiliserons une architecture distribuée sans annuaire reposant sur une stratégie d'inondation contrôlée. Notre proposition devra être adaptée aux spécificités des flottes autonomes décrites en section 5.1.

CHAPITRE 6

Du système au modèle

Sommaire

6.1 Principe général

Dans ce chapitre, nous proposons un nouveau mécanisme distribué appelé AMiRALE (*Asynchronous Missions Relay for Autonomous and Lively Entities*) qui permet à une flotte autonome fortement dynamique de réaliser un ensemble de tâches de manière collaborative. Ce mécanisme repose sur deux notions fondamentales : les *événements* et les *missions*.

Définition 6.1. *Un **événement** est le déclencheur de la création d'une tâche que la flotte doit effectuer.*

Un événement est constitué d'un ensemble de conditions qui peuvent être de nature variée : faible niveau de batterie, seuil de température dépassé, mouvement suspect, etc. Un événement est détecté lorsque l'ensemble des conditions qui le définissent sont remplies. Chaque événement est caractérisé par un *type e* qui permet de

définir la nature de l'action induite que la flotte doit effectuer. Ce type e représente la nature de l'événement et ne doit donc pas être confondu avec une instance de cet événement.

Définition 6.2. *Une **mission** est la description d'une tâche élémentaire que la flotte doit effectuer, générée suite à la détection d'un événement.*

Une mission est toujours liée à un unique type d'événement. En d'autres termes, un événement de type e conduit toujours à la création d'une mission de type e. Dans la suite de ce document, les terme *type* se rapportera donc à un type d'événement ou à un type de mission, ces notions étant équivalentes. Pour tout type e, chaque entité de la flotte possède un rôle particulier : *capteur*, *solveur* ou *relais*.

Définition 6.3. *Une entité dite **capteur** de e ($Sens_e$) est capable de détecter un événement de type e et de créer une mission associée.*

Définition 6.4. *Une entité dite **solveur** de e ($Solv_e$) est capable d'effectuer la tâche relative à une mission de type e.*

Définition 6.5. *Une entité dite **relais** de e ($Forw_e$) est une entité qui n'est ni capteur de e, ni solveur de e mais qui est néanmoins capable de relayer les informations relatives aux missions de type e aux autres membres de la flotte.*

Pour un même type e, un nœud peut à la fois être capteur et solveur. Ceci n'est néanmoins pas toujours possible en raison des limitations physiques des engins (*e.g.* le poids, la taille, le coût). Un nœud peut avoir des rôles distincts pour des types différents de mission. En effet, pour trois types d'événement distincts x, y et z, un nœud avoir les rôles $Sens_x$, $Solv_y$ et $Forw_z$.

Les interactions entre ces éléments sont illustrées à la figure 6.1 au travers d'un scénario d'incendie. Les étapes de ce scénario sont les suivantes :

1. un nœud capteur détecte un incendie ;
2. ce capteur crée une mission relative à cet événement ;
3. la mission est transmise à un solveur par l'intermédiaire de nœuds relais ;
4. un solveur récupère la mission et la résout, par exemple dans ce cas, en effectuant une communication longue distance vers un centre de commandement.

6.2 Missions

6.2.1 Description d'une mission

Lors de de la détection d'un événement de type e, un nœud $Sens_e$ génère une nouvelle mission $m_e^{n:k}$ où n est l'identifiant du créateur de la mission et k est un numéro de séquence local au nœud n, incrémenté à chaque nouvelle mission créée. Le triplet $\{e, k, n\}$ représente l'identifiant unique de la mission. La mission $m_e^{n:k}$ est décrite par un 7-uplet $\{e, k, n, t, s, n', t'\}$ où t est la date de création de la mission,

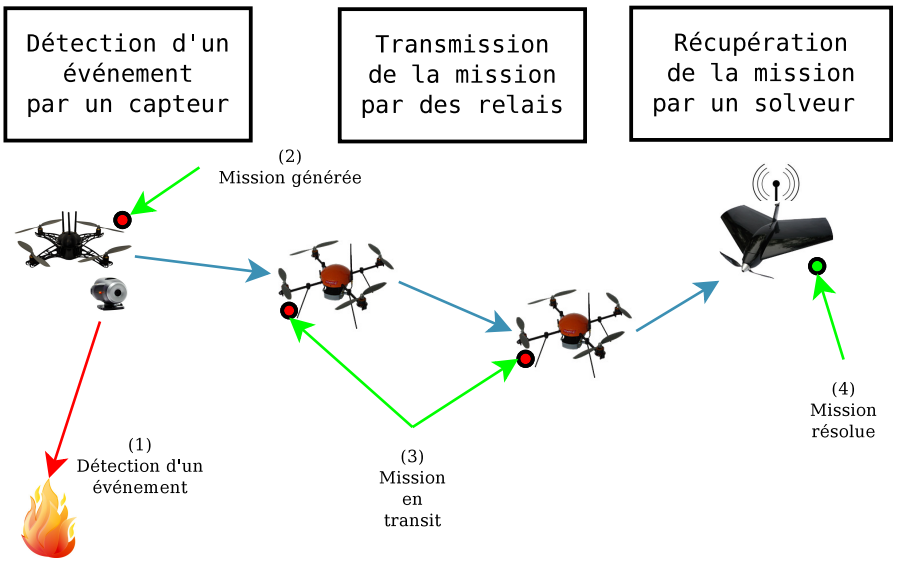


FIGURE 6.1 – Illustration d'AMiRALE dans le cadre d'un scénario d'incendie.

s représente l'état courant de la mission [1], n' est l'identifiant du nœud qui a mis à jour la mission le plus récemment et t' est la date de cette mise à jour. Nous considérons que chaque nœud de la flotte possède un identifiant unique ainsi que sa propre horloge interne [2].

Une mission contient généralement un ensemble de données associées. Par exemple, une mission générée suite à la détection d'un incendie devrait contenir la position du foyer, la température détectée ou encore l'étendue de l'incendie. Nous attirons l'attention du lecteur sur le fait que ces données ne sont pas représentées dans notre modèle même si elles sont bien contenues dans chacune des missions. Nous considérons par ailleurs que les données relatives à une même mission sont immuables pendant la totalité du cycle de vie de la mission.

6.2.2 Diffusion d'une mission

À intervalle de temps régulier, les nœuds diffusent à leurs voisins tout ou partie des missions dont ils ont connaissance. Cela permet aux nœuds d'échanger et de mettre à jour les versions des différentes missions que la flotte doit effectuer. Nous appelons *ensemble de missions*, la totalité des missions connues par un même nœud. Une mission $m_e^{n:k}$ est diffusée sous la forme d'une *vue*, notée $v_e^{n:k}$, qui est une forme réduite de cette mission [3]. En d'autres termes, $m_e^{n:k}$ est la vision locale à un nœud d'une mission et $v_e^{n:k}$ est sa version réduite extraite et utilisée lors des communications avec les autres nœuds de la flotte.

Les communications entre les nœuds sont exclusivement réalisées par *broadcast*. Les messages ne sont pas acquittés ; les nœuds se contentent donc de diffuser les vues des missions dont ils ont connaissance et de prendre en compte les vues des

1. Les états possibles d'une mission seront décrits dans la section suivante.
2. La problématique de la synchronisation entre les horloges sera traitée dans la section 6.4.
3. Le contenu des vues sera détaillé dans la section 6.4.

missions qu'ils reçoivent des autres nœuds de la flotte. Ce fonctionnement *asynchrone* assure la tolérance à une forte mobilité. En effet, toutes les communications étant unidirectionnelles et sans multi-saut, elles ne nécessitent pas de synchronisation entre les nœuds. De plus, la diffusion par *broadcast* permet de ne pas recourir à l'utilisation de protocoles de routage sur le réseau sous-jacent.

Afin de réduire la quantité de messages émis, l'ensemble des vues devant être diffusées par un nœud sont agrégées en un message unique. Ce message est appelé *méta-vue*. Une méta-vue contient toujours l'identifiant de son émetteur et éventuellement un ensemble de données supplémentaires. Ces données peuvent par exemple être utilisées pour diffuser l'état de l'émetteur du message (*e.g.* niveau de batterie, position géographique).

6.2.3 États d'une mission

Un des éléments de la description de la mission $m_e^{n:k}$ est l'état interne s. Celui-ci évolue au cours de la vie de la mission et peut prendre cinq valeurs distinctes :

1. *start* : la mission a été créée mais aucun solveur ne tente pour le moment de la résoudre.
2. *will* : la mission a été récupérée par un solveur qui se prépare à la résoudre. Cette préparation est notamment utile lorsqu'un solveur doit se déplacer vers un lieu précis pour résoudre une mission.
3. *do* la mission est en cours de résolution par un solveur.
4. *abort* : la mission a été abandonnée car elle a été considérée comme invalide. Une mission est dite invalide lorsque l'objectif visé ne peut pas être rempli. Par exemple, un solveur dont la mission est de ramasser un objet situé à une position précise considérera sa mission comme invalide si l'objet recherché n'est pas trouvé. Dans le cadre d'un incendie, une entité chargée de larguer l'agent extincteur (*e.g.* du gaz, de l'eau, de la poudre, de la mousse) considérera sa mission comme invalide si le feu n'est plus actif. L'état *abort* ne correspond donc pas à une erreur de traitement ou à une panne du solveur. Cet état ne peut être atteint que si le solveur dispose des capacités nécessaires lui permettant de considérer sa mission comme invalide.
5. *end* : la mission est résolue.

Pour un type de mission e donné, l'état *start* ne peut être affecté que par un capteur de e au moment de la création d'une mission de même type. Les autres états ne peuvent être assignés à une mission $m_e^{n:k}$ que par des solveurs de e. Les relais de e sont par nature des nœuds fonctionnant en lecture-seule (*i.e.* ils ne peuvent en aucun cas modifier l'état d'une mission de type e).

Les états sont strictement ordonnés : $start < will < do < abort < end$. En d'autres termes, pour une mission donnée, il n'est pas possible de revenir dans un état inférieur.

6.2.4 Cycle de vie d'une mission

Lors de la détection d'un événement de type e par un capteur $Sens_e$, ce dernier crée une mission $m_e^{n:k}$ où n est l'identifiant unique du capteur et k est un numéro de séquence local incrémenté à chaque nouvelle mission créée par ce capteur. Cette mission est ensuite diffusée sous forme de vue par le capteur.

Lorsqu'un nœud reçoit la vue d'une mission dont il ne connaît pas l'existence, il l'ajoute à son ensemble de missions et la diffuse à son tour. Dans le cas où un nœud reçoit une version plus récente d'une mission dont il connaît l'existence, il met cette dernière à jour dans son ensemble de missions et en diffuse la nouvelle version.

Lorsqu'un $Solv_e$ a connaissance d'une mission $m_e^{n:k}$ dont l'état est *start*, il modifie cet état en *will* et se prépare à résoudre la mission (*e.g.* en se déplaçant vers une position précise si nécessaire). Quand il commence à résoudre cette mission, il fait évoluer son état en *do*. Une mission dont l'état est *do* ou *will* est dite verrouillée. Dans la suite de ce document, nous utiliserons le terme *verrou* pour spécifier qu'une mission se trouve dans l'état *will* ou l'état *do*. Une mission représentant une tâche élémentaire, celle-ci ne doit être verrouillée que par un seul solveur à un instant donné. On appelle ce principe *unicité du verrou*.

Si le solveur détecte que la mission n'est plus réalisable (*e.g.* un objet devant être ramassé à disparu), il affecte l'état de la mission à *abort*. Enfin, quand le solveur estime que la mission est terminée, il affecte l'état relatif à *end*.

Nous considérons que les entités de la flotte sont mono-tâche ; *i.e.* elles ne peuvent résoudre qu'une seule mission à la fois. Par conséquent, un solveur ne peut être engagé en *will* ou en *do* que pour une seule mission à un instant donné.

De plus, un solveur ne peut conserver l'état *will* (resp. *do*) que pour un certain intervalle de temps Ψ^e_{will} (resp. Ψ^e_{do}). Ces seuils de durée sont spécifiques à un type de mission. Par conséquent, pour deux types distincts e et f, les valeurs des seuils Ψ^e_{will}, Ψ^f_{will}, Ψ^e_{do} et Ψ^f_{do} peuvent être différentes. Ceci permet de limiter indépendamment la durée de traitement de chaque type de mission. Au-delà de ce seuil, le solveur abandonne la mission s'il est informé qu'un autre solveur a pris la main sur la mission concernée. En effet, les informations temporelles (*i.e.* t et t') contenues dans les missions permettent aux nœuds de la flotte d'estimer l'âge d'un verrou sur une mission particulière. Ce mécanisme permet notamment à un autre solveur de re-verrouiller une mission dont le solveur courant fait face à des difficultés de traitement ou à une panne. Lors de cette phase de reprise, la mission est verrouillée par deux solveurs distincts contrevenant ainsi au principe d'unicité du verrou. Le système va donc mettre en œuvre un mécanisme permettant au premier solveur d'abandonner son verrou. Cet aspect permet d'assurer à notre système la tolérance aux pannes ainsi qu'à la volatilité des nœuds de la flotte.

Enfin, si un solveur est informé qu'une mission dans laquelle il est engagé a évolué vers un état supérieur (au sens de l'ordre défini à la section 6.2.3), il l'abandonne et met à jour son ensemble de missions.

L'automate fini représenté en figure 6.2 décrit l'évolution de l'état d'une mission au cours de sa vie et au sein d'un même nœud.

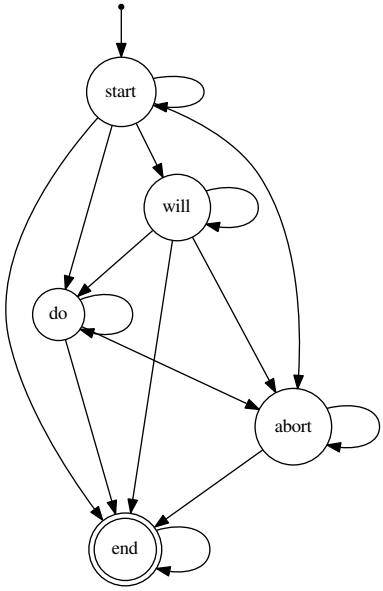


FIGURE 6.2 – Automate fini représentant l'évolution de l'état d'une mission dans la mémoire d'un même nœud.

6.3 Paramètres des types d'événement et de mission

Certaines décisions internes du mécanisme collaboratif sont dépendantes du scénario. Par exemple, créer ou non une nouvelle mission suite à un événement déjà considéré, diffuser ou non une mission terminée depuis un certain temps, sont des décisions qui dépendent entièrement de l'application. C'est pourquoi AMiRALE inclut un certain nombre de paramètres, appelés *variables* et *filtres*, qui permettent de personnaliser et de configurer finement les mécanismes de décision internes du système. De plus, ces configurations peuvent être différentes en fonction de la nature des événements. L'ensemble des paramètres sont donc spécifiques à chaque type.

6.3.1 Variables d'un type

Pour chaque type de mission e, trois variables sont définies par l'application :

— Ψ^e_{will} est l'intervalle de temps pendant lequel un solveur est en droit de conserver son verrou en *will* sur une mission de type e. C'est également la durée pendant laquelle aucun autre solveur connaissant l'état actuel de cette mission ne tentera de reprendre le verrou sur celle-ci. Une fois cet intervalle de temps écoulé, le solveur abandonne la mission si et seulement s'il est informé qu'un autre solveur l'a re-verrouillée. En effet, un solveur ne peut relâcher une mission en *will* que dans les cas suivants :
  — le solveur fait passer la mission dans l'état *do*, *abort* ou *end* ;
  — le solveur est informé que la mission a évolué dans un état supérieur (*i.e.* *do*, *abort* ou *end*) par l'action d'un autre solveur ;

— le solveur dispose du verrou sur la mission depuis une durée supérieure à Ψ^e_{will} et il est informé que la mission a été verrouillée par un autre solveur ;
— le solveur dispose du verrou sur la mission depuis une durée inférieure à Ψ^e_{will} et il est informé que la mission a été verrouillée par un autre solveur dont la *pertinence* est supérieure [4]. Dans le cas où deux solveurs ne sont pas en mesure de communiquer et qu'ils possèdent en mémoire la même mission dans l'état *start*, ils peuvent verrouiller la même mission. Le système va alors mettre en œuvre un mécanisme permettant à un unique solveur de conserver son verrou afin de rétablir le principe d'unicité du verrou définit en section 6.2.4. [5].

— Ψ^e_{do} est l'intervalle de temps pendant lequel un solveur est en droit de conserver son verrou en *do* sur une mission de type e. Les précisions apportées vis à vis du seuil Ψ^e_{will} sont également applicables pour la variable Ψ^e_{do}.
— Φ^e représente le *rang* du type e ; il détermine sa priorité de traitement. Lorsque plusieurs missions de types différents sont disponibles pour verrouillage et que le solveur n'est pas occupé (*i.e.* il ne dispose d'aucun verrou sur aucune mission), le rang permet de sélectionner la mission la plus urgente.

6.3.2 Filtres d'un type

Les *filtres* permettent d'adapter le comportement de certaines parties du modèle afin de répondre aux spécificités du scénario. Dans l'implémentation du modèle, les filtres sont matérialisés par des fonctions utilisateurs. Tout comme c'est le cas pour les variables, chaque type possède ses propres filtres. Nous décrivons ici l'ensemble des filtres disponibles.

Le filtre $f^e_{ignore}(m^{n:k}_e)$ est utilisé lors de la détection d'un événement de type e par un capteur $Sens_e$ dont l'identifiant est n. Ce filtre permet de décider si la mission $m^{n:k}_e$ nouvellement créée suite à cette détection doit être supprimée. Ce filtre permet notamment d'empêcher la création d'une nouvelle mission en réponse à un événement déjà en cours de traitement. Prenons l'exemple d'un type d'événement qui consiste à détecter la présence de mauvaises herbes sur une surface agricole. Il n'est pas nécessaire pour le capteur n de recréer une nouvelle mission relative à une zone contenant cette mauvaise herbe si celle-ci fait déjà l'objet d'une mission en cours de traitement. Ce filtre permet donc de limiter le nombre de missions dont l'objet serait identique.

Le filtre $f^e_{pass}(m^{n:k}_e)$ empêche un solveur de verrouiller la mission $m^{n:k}_e$. Cette fonction est notamment utile pour prendre en compte l'état actuel du solveur dans le choix de la résolution d'une mission. Pour illustrer l'utilisation de ce filtre, nous prenons l'exemple d'un type de mission qui consiste à transporter un objet vers une zone prédéfinie. Dans ce contexte, il est pertinent de prendre en compte le niveau d'autonomie de la batterie de l'entité afin de lui éviter de sélectionner une mission dont la consommation énergétique requise dépasserait la charge disponible. Cette fonction permet également d'intégrer un mécanisme de sélection de missions (i.e. un mécanisme de sélection de tâches comme ceux décrits dans le chapitre 5). Pour

4. La notion de pertinence sera abordée dans la section 6.3.2.
5. Ce point sera également traité en section 6.3.2.

le même scénario de transport d'objet, une stratégie pourrait par exemple être de sélectionner l'objet qui se trouve au plus près de l'entité.

Le filtre $f^e_{blind}(m^{n:k}_e)$ permet d'empêcher la diffusion d'une mission. Ce filtre peut permettre de réduire le volume de trafic réseau généré et donc de limiter les risques de collision et de congestion décrits dans la section 2.3. Ce filtre permet d'implémenter les différentes stratégies d'inondation présentées dans la section 2.3.2.

Le filtre $f^e_{check}(src : v^{n:k}_e)$ permet d'ignorer la vue $v^{n:k}_e$ émise par le nœud dont l'identifiant est *src*. Cette fonction peut par exemple permettre d'implémenter un mécanisme de vérification ou de politique de sécurité. En effet, certains nœuds peuvent ne pas être autorisés à diffuser certains types de mission. Cette interdiction peut être temporaire ou permanente. Il se peut également que certains solveurs ne soient pas autorisés à verrouiller certains types de mission. Ce filtre permet donc d'intégrer des mécanismes de liste noire et de liste blanche. Ce filtre permet également d'intégrer un système d'authentification à base de signatures permettant ainsi de ne pas prendre en compte les vues émises par des entités ne faisant pas partie de la flotte. Les signatures sont dans ce cas intégrées dans les vues comme données supplémentaires (*c.f.* section 6.2.2).

Le filtre $f^e_{select}(self : m^{n:k}_e, src : v^{n:k}_e)$ permet au solveur *self* de déterminer localement s'il doit abandonner le traitement de la mission $m^{n:k}_e$ au profit du solveur *src* qui a diffusé la vue $v^{n:k}_e$ et qui est également en cours de traitement sur la même mission $m^{n:k}_e$. Ce filtre permet ainsi de gérer les conflits sur une même mission entre solveurs. En effet, la mobilité de la flotte et la sporadicité des communications peuvent amener deux solveurs à verrouiller la même mission contrevenant ainsi au principe d'unicité du verrou (*i.e.* une mission ne doit être verrouillée que par un seul solveur à un instant donné). Ce filtre permet de corriger ce phénomène en effectuant une sélection entre les solveurs *self* et *src*. Les critères permettant d'effectuer cette sélection sont dépendant du scénario et du type de mission concerné. Le nœud ainsi sélectionné est considéré comme le plus *pertinent* dans le cadre de la mission concernée. Par exemple, le ramassage d'un objet situé à une position précise pourrait être réalisé par l'entité la plus proche géographiquement de l'objet visé. L'utilisation de ce filtre ainsi que des précisions sur le mécanisme de sélection locale seront détaillées dans le chapitre 7.

6.4 Synchronisation temporelle des missions

Chaque nœud possède sa propre horloge et chacune peut être configurée et/ou dériver d'une façon différente pour un certain nombre de raisons telles que la température ou à cause des propriétés du quartz utilisé [218]. Notre approche étant basée sur des seuils temporels pour le traitement des missions (*i.e.* Ψ^{will} et Ψ^{do}), ces dérives peuvent conduire à des incohérences. Prenons par exemple le cas d'une mission $m^{n:k}_e$ verrouillée par le solveur x depuis 5 secondes et considérons que le seuil de traitement Ψ^{will}_e est fixé à 15 secondes. Un autre solveur de e disponible (*i.e.* sans verrou pour le moment) appelé y est mis au courant de l'existence de la mission $m^{n:k}_e$ au travers de la vue $v^{n:k}_e$ émise par le solveur x. Si l'horloge de y avance de 10

secondes par rapport à celle de x, le solveur y considérera que la mission $m_e^{n:k}$ est verrouillable car le nœud x n'aura pas su résoudre la mission dans le temps Ψ_e^{will} imparti. Cet exemple démontre la nécessité de prendre en compte le problème de la dérive d'horloge au sein de notre mécanicisme collaboratif.

Il existe principalement trois approches qui permettent de lutter contre les effets de ce phénomène [219] :

- L'*approche par temps absolu* (*global timing*) permet de synchroniser de manière globale l'ensemble des horloges présentes dans le réseau. Cette approche est notamment utilisée par le protocole NTP [220] ainsi que par la technologie GPS [221].
- L'*approche par temps relatif* (*relative timing*) permet de prendre en compte la dérive d'horloge entre voisins en embarquant des informations de synchronisation (*i.e.* un *offset*) dans les messages afin d'ajuster les valeurs des dates transmises [222].
- L'*approche par ordre relatif* (*relative ordering*) permet d'ordonner partiellement des événements, informations ou messages sans pour autant effectuer de synchronisation temporelle.

Le protocole NTP repose sur la présence d'un serveur NTP qui diffuse régulièrement la valeur de son horloge aux autres entités du réseau pour que ces dernières corrigent les leurs. De par la nature distribuée des flottes autonomes, le serveur NTP doit être hébergé sur un des nœuds de la flotte. Il est également possible d'utiliser plusieurs serveurs qui doivent dans ces conditions se synchroniser entre eux. Le caractère fortement dynamique des flottes auxquelles nous nous intéressons ne permettrait pas de disposer en permanence de communications entre les serveurs NTP et les autres entités du réseau. De plus, les pannes éventuelles des nœuds hébergeant les serveurs NTP pourraient porter atteinte à l'intégrité de ce mécanisme. L'utilisation d'un tel système n'est donc pas possible dans le cas d'une flotte autonome fortement dynamique.

Le GPS est une technologie de positionnement permettant de synchroniser les horloges des entités qui l'utilisent avec une précision de l'ordre de la nanoseconde [221]. Le réseau GPS étant déployé à échelle mondiale par l'utilisation d'un ensemble de satellites, cette solution semble être la plus pertinente. Néanmoins, cette technologie n'a pas été prévue pour fonctionner en intérieur. De plus, en extérieur, elle est très sensible à l'environnement ; il a par exemple été prouvé que la végétation ainsi que les reliefs pouvaient engendrer des pertes de communication entre les satellites GPS et les clients qui l'utilisent [223]. La technologie GPS ne garantit donc pas une synchronisation temporelle permanente. Enfin, certaines entités de la flotte peuvent ne pas être équipées d'un tel système en raison de son coût énergétique élevé [224].

Pour ces raisons nous avons choisi de développer trois versions de notre système AMiRALE, répondant chacune à une des stratégies de synchronisation temporelle évoquées ci-dessus. Ces versions diffèrent donc principalement par le contenu des vues utilisées pour diffuser et mettre à jour les missions dans la flotte. D'un point de vue technique, comme nous l'avons vu dans la section 6.2.1, chaque mission contient la date de sa création ainsi que celle de sa dernière mise à jour. Ces informations ne peuvent être transmises telles quelles que dans le cas où les horloges sont synchronisées (*i.e.* avec l'approche par temps absolu). Dans les autres cas, des

adaptations doivent être opérées afin d'assurer la cohérence du système. Nous détaillons ci-dessous le contenu des vues en fonction de l'approche de synchronisation temporelle utilisée :

— Dans l'approche par temps absolu, aucune modification n'est nécessaire dans la mesure où les informations temporelles contenues dans les missions sont cohérentes pour l'ensemble des entités de la flotte. Par conséquent, $m_e^{n:k} = v_e^{n:k}$ dans cette approche.

— Dans l'approche par temps relatif, les vues contiennent des intervalles de temps (*i.e.* des *offset*) plutôt que des temps absolus. Pour une mission $m_e^{n:k}$ représentée par le 7-uplet $\{e, k, n, t, s, n', t'\}$, la vue relative $v_e^{n:k}$ sera exprimée par $\langle e, k, n, \Delta, s, n', \Delta' \rangle$ où $\Delta = \Gamma - t$ et $\Delta' = \Gamma - t'$, Γ représentant la valeur actuelle de l'horloge du nœud émetteur de la vue $v_e^{n:k}$. Nous considérons ici que cette valeur est stockée sous forme d'entier représentant la durée écoulée depuis un point d'origine prédéfini, le plus courant étant le temps UNIX (*i.e.* 01/01/1970 à 0H00 UTC). Cette technique permet aux nœuds de partager des informations temporelles représentées par des intervalles de temps et non par des dates absolues. Cette approche peut néanmoins introduire de légers décalages car elle ne prend pas en compte le temps de transmission des vues entre les nœuds ni les faibles différences de vitesse qui peuvent exister entre les horloges [224].

— Dans l'approche par ordre relatif, aucune information temporelle n'est partagée entre les nœuds de la flotte. Ainsi, la vue $v_e^{n:k}$ de la mission $m_e^{n:k}$ est exprimée par le 5-uplet $\langle e, k, n, s, n' \rangle$. Dans cette approche, les mesures de durée pour les verrous sont calculées par rapport à la date d'intégration de la mission dans l'ensemble de missions de chaque nœud et non par rapport à une date fixée. Cette stratégie implique une fréquence de mise à jour des missions moins importante pouvant ainsi conduire à une augmentation des durées de leur traitement.

Les vues étant agrégées dans des méta-vues contenant également des données supplémentaires, il est possible d'y ajouter le type de synchronisation utilisé dans ces vues. Il est ainsi possible de mettre en place un mécanisme permettant de changer de stratégie en fonction de la qualité de synchronisation des horloges.

6.5 Résumé

Dans ce chapitre, nous avons présenté les principes ainsi que le fonctionnement général de notre mécanisme collaboratif appelé AMiRALE, basé uniquement sur des opérations locales et sur des messages asynchrones. Après avoir détaillé les différentes notions ainsi que les composants utilisés dans notre approche, nous avons introduit les différents paramètres permettant d'adapter le comportement du modèle aux spécificités du scénario. Nous avons ainsi défini et détaillé le principe des *filtres* qui est au cœur de notre système. Enfin, nous avons abordé le problème de la synchronisation des horloges dans un réseau mobile de type MANet et avons proposé une adaptation de notre système pour chaque solution existante.

CHAPITRE 7

Modélisation d'AMiRALE

Sommaire

7.1 Approches par ré-étiquetage de graphes

Afin de formaliser les opérations locales ainsi que les communications effectuées par les nœuds dans AMiRALE, nous avons choisi d'utiliser le ré-étiquetage de graphes [225]. Ce modèle permet de représenter les opérations qui sont faites localement sur les nœuds en tenant compte uniquement de l'état de leurs voisins, de

leur propre état, ainsi que des liens de communication existants. Dans ce contexte, les sommets du graphes représentent les nœuds du réseau et les arêtes leurs canaux de communication. Il a été démontré que les algorithmes décrits sous forme de ré-étiquetage de graphes pouvaient être implémentés par des systèmes distribués reposant uniquement sur des échanges de messages asynchrones [226]. Ce modèle est donc adapté à AMiRALE. Nous détaillons dans cette section trois évolutions du modèle de ré-étiquetage de graphes qui nous permettront de formaliser AMiRALE.

7.1.1 Modèle statique

GRS (*Graph Relabelling System*) est un modèle de ré-étiquetage de graphes statiques [227]. Dans ce modèle on considère que les arêtes et les sommets présents dans le graphe ne peuvent jamais disparaître. De même, aucun sommet et aucune arête ne peuvent apparaître au cours du scénario. Dans ce modèle, chaque sommet ou arête possède une étiquette qui peut être modifiée lors de l'application d'une règle de ré-étiquetage. La figure 7.1 montre un exemple de ces règles. Dans cet exemple, la règle indique que deux sommets respectivement étiquetés A et B et reliés par une arête étiquetée i appliqueront la règle donnée en changeant leurs étiquettes respectivement en C, D, et j.

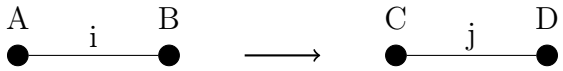


FIGURE 7.1 – Exemple d'une règle de ré-étiquetage du modèle GRS.

Jérémie Chalopin a défini quatre types distincts d'interaction pouvant intervenir dans un tel système de ré-étiquetage [228]. Ces types d'interaction sont illustrés à la figure 7.2. Ils peuvent être caractérisés comme ci-dessous :

- L'interaction LC_2 (figure 7.2a), dite *en étoile fermée*, permet de modifier l'état du nœud qui exécute le calcul en tenant compte de l'état de l'ensemble de ses voisins ainsi que de son propre état. Lors de cette opération, les états de l'ensemble des voisins du nœud qui exécute le calcul peuvent être modifiés.
- L'interaction LC_1 (figure 7.2b), dite *en étoile ouverte*, permet de modifier que l'état du nœud qui exécute le calcul mais en tenant tout de même compte de l'état de chacun de ses voisin en plus de son propre état.
- L'interaction LC_0 (figure 7.2c), dite *en paire fermée*, permet de modifier l'état de deux nœuds voisins en tenant compte de leurs états respectifs.
- L'interaction LC_0^{cell} (figure 7.2d), dite *en paire ouverte*, permet de modifier l'état du seul nœud effectuant le calcul mais en tenant compte de l'état d'un de ses voisins et de son propre état.

L'ensemble de ces types d'interaction suppose des échanges entre le nœud exécutant le calcul et un ou plusieurs de ses voisins.

Le modèle GRS n'a pas été conçu pour formaliser des algorithmes évoluant sur des réseaux dynamiques et volatiles. Par conséquent, ce modèle n'est pas adapté au contexte de notre étude. C'est pourquoi nous nous intéressons à une adaptation dynamique de ce modèle dans la section suivante.

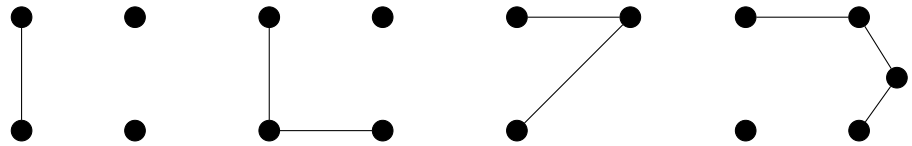
(a) LC_2 (b) LC_1

(c) LC_0 (d) LC_0^{cell}

FIGURE 7.2 – Illustration des différents types d'interaction du modèle GRS.

7.1.2 Modèle dynamique

DAGRS (*Dynamicity Aware Graph Relabeling System*) est une évolution du modèle GRS reposant sur des graphes dynamiques [229, 230]. Un graphe dynamique est un graphe dont les arêtes et les nœuds peuvent disparaître à tout moment. De plus, de nouveaux nœuds ainsi que de nouvelles arêtes peuvent faire leur apparition au cours du scénario. Un graphe dynamique G peut être représenté sous la forme d'une succession de graphes statiques G_i où chaque G_i représente l'état du réseau à l'instant i. Dans ce modèle, le temps est donc discrétisé. Une illustration d'un tel graphe est proposée à la figure 7.3.



(a) G_1 (b) G_2 (c) G_3 (d) G_4

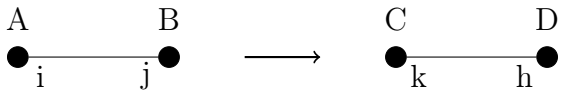
FIGURE 7.3 – Illustration de l'évolution d'un graphe dynamique sur quatre périodes.

Le modèle DAGRS adapte le modèle GRS à un contexte dynamique de la façon suivante :

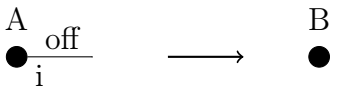
— Les étiquettes d'arêtes sont codées sur leurs brins. L'arête a donc une étiquette à chacune de ses extrémités, étiquettes qui peuvent être différentes. Cette approche permet d'introduire des états asymétriques sur les arêtes.
— Deux types de règles correspondant à l'apparition et à la disparation d'une arête (*i.e.* lien de communication) sont ajoutées au modèle.

Les types de règles de ré-étiquetage du modèle DAGRS sont illustrées à la figure 7.4.

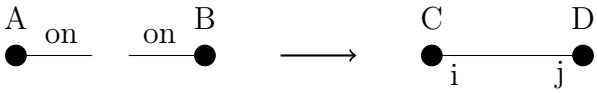
Bien que le modèle DAGRS soit conçu pour être utilisé sur des graphes dynamiques et donc des réseaux mobiles, les types de règles qu'il propose nécessitent d'effectuer des synchronisations entre les nœuds. Il fonctionne en effet avec des interactions de type LC_0. Cet aspect va à l'encontre du fonctionnement de notre système AMiRALE qui n'utilise que des communications asynchrones et unidirectionnelles. C'est la raison pour laquelle, nous nous intéressons à une adaptation asynchrone de ce modèle dans la section suivante.



(a) Règle classique.



(b) Règle de disparition d'une arête.



(c) Règle d'apparition d'une arête.

FIGURE 7.4 – Types de règles de ré-étiquetage du modèle DAGRS.

7.1.3 Modèle dynamique asynchrone

ADAGRS (*Asynchronous Dynamicity Aware Graph Relabeling System*) est une adaptation du modèle DAGRS reposant exclusivement sur des échanges de messages asynchrones [72, 231]. Pour cela, ce modèle introduit un nouveau type d'interaction appelé LC_0^{async} qui est une version asynchrone des interactions en paire ouverte (*i.e.* LC_0^{cell}). LC_0^{async} est illustré à la figure 7.5. Pour appliquer ce type de règle de ré-étiquetage, le nœud effectuant le calcul utilise la dernière information disponible sur l'état du nœud voisin considéré dans l'opération. Ainsi, ce type d'interaction ne suppose pas de synchronisation entre les deux nœuds de la paire impliquée dans la règle de ré-étiquetage. Les communications considérées étant asymétriques, l'arête présente dans la représentation de LC_0^{async} est orientée.

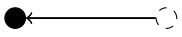
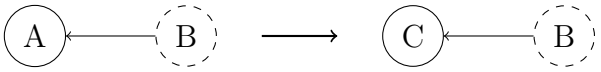


FIGURE 7.5 – Illustration du type d'interaction LC_0^{async}.

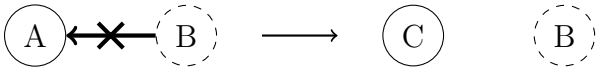
Le modèle ADGRS propose les trois types de règles présents dans DAGRS : classique, disparition et apparition d'une arête. Une illustration de ces types règles adaptés à LC_0^{async} est donnée à la figure 7.6.

Certaines décisions étant dépendantes de l'application, le modèle ADAGRS introduit la notion d'*événement applicatif*. Ces événements sont déclenchés par l'application utilisant le modèle ADAGRS afin d'appliquer des règles spécifiques. La figure 7.7 illustre le prise en compte de ces événements dans le modèle.

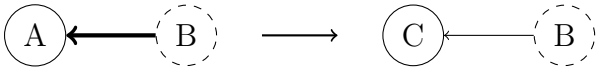
Le modèle ADAGRS inclut également un mécanisme d'échange de messages asynchrones dont le fonctionnement est illustré à la figure 7.8.



(a) Règle normale.



(b) Règle de disparition d'une arête.

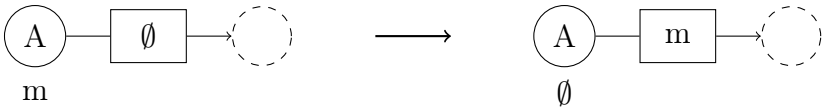


(c) Règle d'apparition d'une arête.

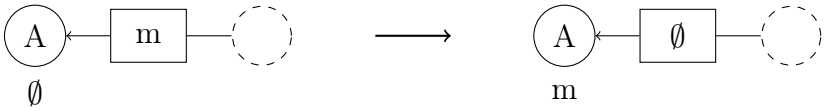
FIGURE 7.6 – Types de règles de ré-étiquetage du modèle ADAGRS.

(a) Événement envoyé par l'application. (b) Réaction du modèle à un événement.

FIGURE 7.7 – Règles relatives aux événements du modèle ADAGRS.



(a) Envoi d'un message.



(b) Réception d'un message.

FIGURE 7.8 – Règles de gestion des messages du modèle ADAGRS.

7.1.4 Discussion

Dans cette section nous avons pu préciser que le modèle GRS n'a pas été conçu pour des réseaux mobiles. C'est pourquoi le modèle DAGRS a été introduit. Ce dernier adapte le modèle GRS et permet ainsi d'être utilisé sur des réseaux dynamiques (*i.e.* mobiles et volatiles). Néanmoins, il nécessite des synchronisations ainsi que des échanges de messages bidirectionnels entre les nœuds pour l'application des règles de ré-étiquetage. Le modèle ADAGRS a été développé dans l'optique de dépasser ces contraintes. Ainsi, il fonctionne entièrement de manière asynchrone et ne nécessite donc pas d'échange de message entre les nœuds pour l'application des règles de ré-étiquetage. Le modèle ADAGRS reposant uniquement sur des opérations locales ainsi que des messages asynchrones, il pourrait être utilisé pour modéliser notre système AMiRALE.

L'élément central d'AMiRALE réside dans l'état des différentes missions que la flotte doit effectuer. Bien que l'état de l'ensemble des différentes missions au sein d'un même nœud pourrait être modélisé grâce à ADAGRS, cette formalisation serait trop complexe à représenter (*i.e.* elle serait peu lisible et très verbeuse).

C'est la raison pour laquelle, nous développons un nouveau formalisme qui s'inspire des principes du modèle ADAGRS.

7.2 Adaptation de l'approche pour AMiRALE

Dans cette section nous présentons le formalisme utilisé pour décrire les opérations effectuées par les nœuds sur les missions au sein d'AMiRALE. Chacune de ces opérations est décrite par une règle illustrée (comme c'est le cas pour les modèles de ré-étiquetage de graphes présentés dans la section précédente). Les différentes règles décrivent les opérations effectuées sur une seule et même mission. Par conséquent, pour une règle donnée, seule la mission concernée est considérée.

Comme nous l'avons expliqué dans le chapitre 6, pour tout type d'événement e, chaque nœud du réseau possède un rôle particulier :

- $Sens_e$ signifie que le nœud peut détecter des événements de type e et créer les missions associées ;
- $Solv_e$ signifie que le nœud peut résoudre une mission de type e ;
- $Forw_e$ signifie que le nœud peut relayer aux autres nœuds de la flotte les informations qu'il connaît sur une mission de type e (rôle par défaut).

Un nœud peut éventuellement être à la fois capteur et solveur pour un même type d'événement. Pour simplifier la description du modèle utilisé, nous définissons un nouveau rôle générique appelé Any_e qui qui représente n'importe lequel des rôles précédemment décrits.

Les règles suivantes décrivent les opérations effectuées par un nœud sur une mission de type e. Par conséquent, pour une règle donnée, seule la mission considérée est représentée et non la totalité des missions portées par le nœud. Pour chacune des règles, les deux cercles représentent le rôle (*i.e.* capteur, solveur, relais ou générique) du nœud avant et après avoir appliqué la règle. L'ensemble en dessous de chaque cercle représente la version courante de la mission $m_e^{n:k}$. Chaque règle est appliquée uniquement si la condition c est satisfaite.

7.2.1 Type de règles relatif aux capteurs

La figure 7.9 représente la détection d'un événement de type e par un capteur dont l'identifiant est n. Ce dernier crée la mission $m_e^{n:k}$ relative à l'événement détecté uniquement si la condition c est satisfaite. Cette condition peut, par exemple, servir à vérifier qu'aucune autre mission relative au même événement n'est déjà en cours de traitement. La création de la mission est annulée dans le cas contraire. Cette règle est la seule qui permette la création d'une nouvelle mission.

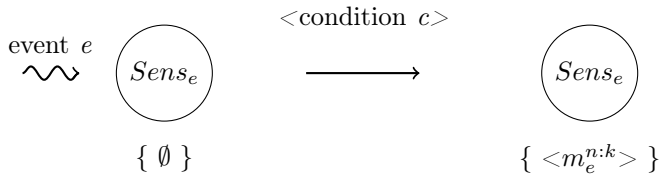


FIGURE 7.9 – Règle de détection d'un événement dans AMiRALE.

7.2.2 Types de règles relatifs aux solveurs

La figure 7.10 représente la modification de l'état d'une mission par un solveur. Cette règle permet à un solveur de modifier la version courante d'une mission si la condition c est respectée. Après application de cette règle, la mission $m_e^{n:k}$ est mise à jour en $m_e'^{n:k}$. Cette règle sera notamment utilisée par les solveurs pour verrouiller les missions qu'ils souhaitent résoudre.

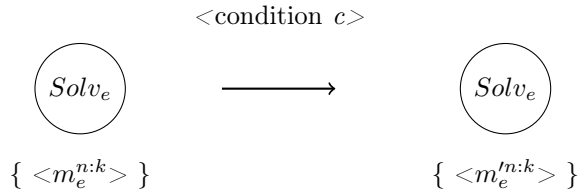


FIGURE 7.10 – Règle d'opération autonome dans AMiRALE.

La figure 7.11 représente la prise en compte d'un événement applicatif par un solveur. Un événement *applicatif* est une remontée d'informations de la part de l'application qui exploite le système AMiRALE vers celui-ci. Ces événements permettent notamment d'indiquer au système collaboratif l'état d'avancement réel dans le traitement d'une mission. AMiRALE permet de gérer la collaboration entre les différents systèmes autonomes qui composent la flotte. Une instance du système AMiRALE est donc active sur chacune de ces entités. En revanche, il n'a pas pour vocation d'être un système indépendant. En effet, AMiRALE dépend d'une application tierce qui est responsable de la gestion des différents capteurs et actionneurs de l'engin sur lequel elle est exécutée. Cette application délègue à AMiRALE les fonctionnalités de prise de décisions. AMiRALE peut donc être considéré comme un outil de supervision. Par conséquent, il doit prendre en considération les informations fournies par l'application qui l'exploite. Prenons l'exemple d'un scénario où un solveur x doit ramasser un objet au sol. Soit la mission $m_e^{n:k}$ relative à cet objectif. Supposons ici que cette mission soit verrouillée en *will* par le solveur x (celui-ci est en train de se diriger vers la position de l'objet visé). Une fois arrivée à destination (*i.e.* une fois la phase de préparation terminée), l'application doit signaler au système AMiRALE qu'elle est en mesure de commencer à effectuer la mission ; AMiRALE fait ainsi passer l'état de la mission $m_e^{n:k}$ en *do*. On notera qu'un événement applicatif est toujours lié à une mission spécifique. La condition c permet d'annuler l'exécution de la règle.

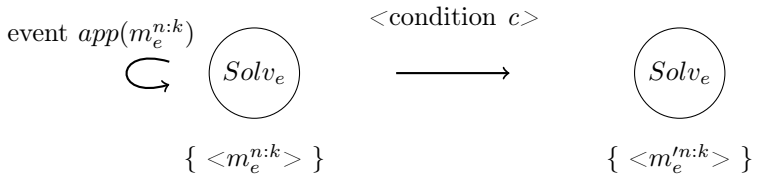


FIGURE 7.11 – Règle relative à un événement applicatif dans AMiRALE.

7.2.3 Types de règles génériques

La figure 7.12 représente la règle d'envoi d'une vue $v_e^{n:k}$ de la mission $m_e^{n:k}$. Les vues étant diffusées par *broadcast*, le message cible ne porte pas de destinataire particulier. En revanche, à chaque vue est associée l'identifiant de son émetteur, représenté ici par *self*. Nous avons également précisé dans le chapitre 6 que plusieurs vues sont agrégées dans un même message (*i.e.* une méta-vue) afin de réduire les risques de saturation réseau liés aux phénomènes de congestion et de collision. Le formalisme que nous utilisons opère individuellement sur chacune des missions, par conséquent l'agrégation n'y est pas représentée. L'envoi d'une vue dans ce formalisme revient à son ajout dans la prochaine méta-vue à envoyer par le nœud concerné par la règle. La condition c permet de décider si la vue concernée par la règle doit être ajoutée à cette prochaine méta-vue. Cette condition pourra notamment être utilisée afin d'intégrer les différentes stratégies d'inondation présentées dans la section 2.3.2.

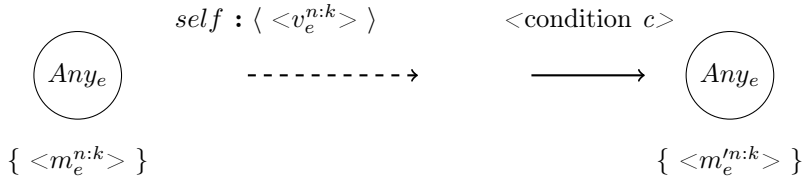


FIGURE 7.12 – Règle d'envoi d'une vue dans AMiRALE.

La figure 7.13 représente la réception de la vue $v_e^{n:k}$ relative à la mission $m_e^{n:k}$, émise par le nœud dont l'identifiant est *src*. Cette règle peut notamment entraîner la mise à jour d'une mission dont la version locale est périmée.

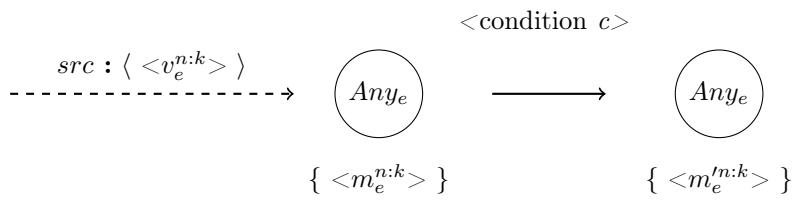


FIGURE 7.13 – Règle de réception d'une vue dans AMiRALE.

7.3 Règles de ré-étiquetage d'AMiRALE

Dans cette section, nous présentons l'ensemble des règles du système AMiRALE reposant sur le formalisme présenté dans la section 7.2. Comme nous l'avons vu dans le chapitre 6, il existe trois versions différentes de notre système reposant chacune sur une stratégie distincte de synchronisation des horloges des nœuds de la flotte. Nous présentons tout d'abord les règles communes à ces approches puis nous détaillons les règles spécifiques à chacune d'entre elles. Ces règles spécifiques portent uniquement sur les communications entre les nœuds. Les règles impliquant uniquement des opérations locales sont communes aux trois versions.

7.3.1 Règles communes

Règle relative aux capteurs

La figure 7.14 représente la détection d'un événement de type e par un capteur dont l'identifiant est $self$. Lors de l'exécution de cette règle, le numéro de séquence k local au nœud $self$ est incrémenté et associé à la mission $m_e^{self:k}$ ainsi créée. Le filtre f^e_{ignore} est utilisé pour déterminer si la création de la mission doit être annulée. Ce filtre peut par exemple être configuré pour empêcher la création de plusieurs missions relatives au même événement. L'état de la mission est fixé à *start*. Le nœud $self$ indique qu'il est le créateur de la mission et utilise la valeur courante de son horloge interne (*i.e.* la notation Γ) pour fixer la date de création de la mission. Les informations de dernière modification sont identique à celles relatives à la création de la mission. Les paramètres soulignés sont ceux qui prennent une nouvelle valeur suite à l'application de la règle.

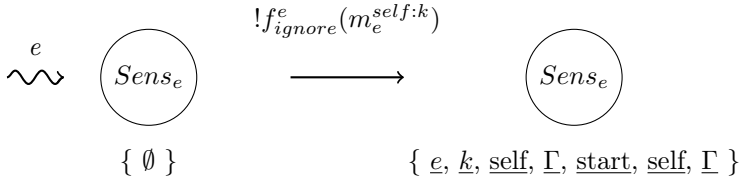
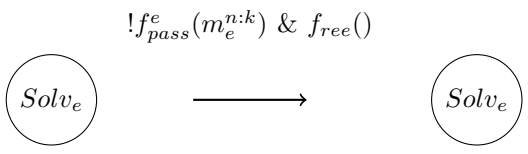


FIGURE 7.14 – Détection d'un événement par un capteur.

Règles relatives aux opérations locales des solveurs

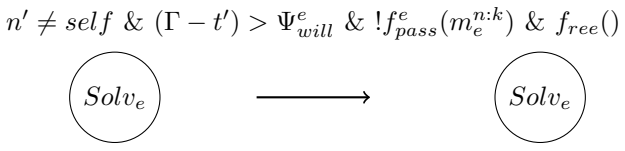
La règle représentée dans la figure 7.15 permet à un solveur de type e de verrouiller en *will* une mission $m_e^{n:k}$ dont l'état est *start*. Cette opération est possible uniquement si le solveur est *libre* (fonction f_{ree}), *i.e.* si aucune mission n'est actuellement verrouillée en *will* ou en *do* par ce dernier. De plus, le filtre f^e_{pass} est utilisé pour déterminer si le verrouillage de la mission peut être effectué. Ce filtre peut par exemple empêcher le solveur de verrouiller une mission qui nécessiterait plus d'énergie qu'il ne lui en reste. Une fois la règle appliquée, l'état de la mission passe en *will* et les informations de dernière modification, avec l'identifiant du solveur et la date de dernière modification, sont mises à jour.



(2) { e, k n, t, start, n, t } { e, k, n, t, will, self, Γ }

FIGURE 7.15 – Verrouillage d'une mission en *will* par un solveur.

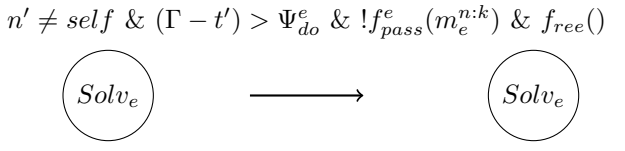
La règle illustrée en figure 7.16 permet à un solveur de type e de re-verrouiller en *will* une mission $m^{n:k}_e$ dont l'état est déjà *will*. Cette règle est notamment utile pour reprendre la main sur une mission verrouillée par un solveur qui aurait rencontré une panne. Les conditions d'application de cette règle sont les même que celles de la règle précédente. De plus, cette règle ne s'applique que si l'intervalle de temps Ψ^e_{will} mesuré depuis le verrouillage de la mission (*i.e.* t') est écoulé. Par ailleurs, un solveur n'est pas autorisé à re-verrouiller une mission sur laquelle il dispose déjà d'un verrou.



(3) { e, k, n, t, will, n', t' } { e, k, n, t, will, self, Γ }

FIGURE 7.16 – Re-verrouillage d'une mission en *will* par un solveur.

La figure 7.17 est similaire à la précédente mais est destinée à une mission verrouillée en *do*. Les conditions d'application sont les mêmes sauf pour ce qui concerne l'intervalle de temps autorisé pour la conservation d'un verrou qui est ici Ψ^e_{do}.

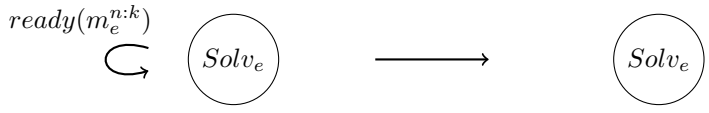


(4) { e, k, n, t, do, n', t' } { e, k, n, t, do, self, Γ }

FIGURE 7.17 – Re-verrouillage d'une mission en *do* par un solveur.

Règles relatives aux événements applicatifs sur des solveurs

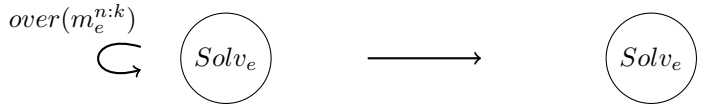
La règle présentée en figure 7.18 permet à l'application d'indiquer à un solveur que la mission sur laquelle il dispose d'un verrou en *will* peut être résolue immédiatement (*i.e.* le temps de préparation est terminé). L'état de la mission passe ainsi à *do*. Cette règle n'est soumise à aucune condition.



(5) { e, k, n, t, will, self, t' } { e, k, n, t, $\underline{\text{do}}$, self, $\underline{\Gamma}$ }

FIGURE 7.18 – Verrouillage d'une mission en *do* par un solveur.

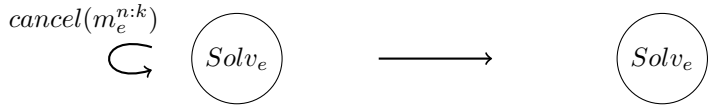
La règle illustrée en figure 7.19 permet à l'application d'indiquer à un solveur que la mission sur laquelle il dispose d'un verrou en *do* est résolue. L'état de la mission passe ainsi à *end*.



(6) { e, k, n, t, do, self, t' } { e, k, n, t, $\underline{\text{end}}$, self, $\underline{\Gamma}$ }

FIGURE 7.19 – Terminaison d'une mission par un solveur.

La règle décrite dans la figure 7.20 permet à l'application d'indiquer à un solveur que la mission sur laquelle il dispose d'un verrou en *do* est invalide. L'état de la mission passe ainsi en *abort*. Pour mémoire, une mission est considérée comme invalide lorsque la tâche associée n'est plus réalisable (*e.g.* un objet devant être ramassé a disparu).



(7) { e, k, n, t, do, self, t' } { e, k, n, t, $\underline{\text{abort}}$, self, $\underline{\Gamma}$ }

FIGURE 7.20 – Annulation d'une mission par un solveur.

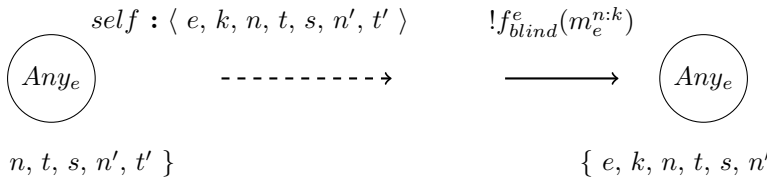
7.3.2 Règles du modèle en temps absolu

Dans l'approche par temps absolu, l'horloge de chaque entité est synchronisée de manière globale, *i.e.* la différence de configuration initiale et de dérive des horloges est ici corrigée. Aucune modification des vues n'est donc nécessaire dans la mesure où les informations temporelles contenues dans les missions sont cohérentes pour l'ensemble des entités de la flotte. Par conséquent, $m_e^{n:k} = v_e^{n:k}$ dans cette approche.

Règle relative à l'envoi d'une vue

La figure 7.21 représente la seule règle qui permet à un nœud de diffuser les vues des missions dont il a connaissance. Nous rappelons que les vues sont transmises par *broadcast* ; le message n'a donc pas de destinataire. En revanche, chaque vue est

associée à l'identifiant de son émetteur. Une vue est diffusée si et seulement si le filtre f^e_{blind} ne retourne pas une interdiction. Ce filtre peut par exemple permettre de réduire le nombre de ré-émissions d'une même vue sur un intervalle de temps donnée. L'émission d'une vue $v^{n:k}_e$ n'implique aucune modification de la mission $m^{n:k}_e$ relative.



(8)

FIGURE 7.21 – Envoi d'une vue en temps absolu.

Règles relatives à la réception d'une vue par un solveur

La règle représentée en figure 7.22 permet à un solveur de relâcher son verrou sur une mission en *will* dans le cas où un autre solveur, nommé ici n', ayant verrouillé la même mission lui communique directement (*i.e.* sans intermédiaire) cette information. Cette décision est prise localement, *i.e.* elle ne nécessite aucune synchronisation entre les deux solveurs impliqués dans l'exécution de la règle. Cette règle est de fait utilisée pour effectuer une élection locale entre deux solveurs ayant verrouillé tous deux la même mission en *will*, l'objectif étant de rétablir le principe d'unicité du verrou défini en section 6.2.4. Cette règle n'est appliquée que si la communication entre les deux solveurs est directe. En d'autres termes, l'élection locale n'est pas effectuée si la vue ne provient pas directement du solveur concerné. Un solveur souhaitant appliquer cette règle va donc comparer l'identifiant du nœud émetteur de la vue reçue et celui associé au verrou indiqué dans cette même vue. Un certain nombre d'autres conditions sont nécessaires à l'application de cette règle :

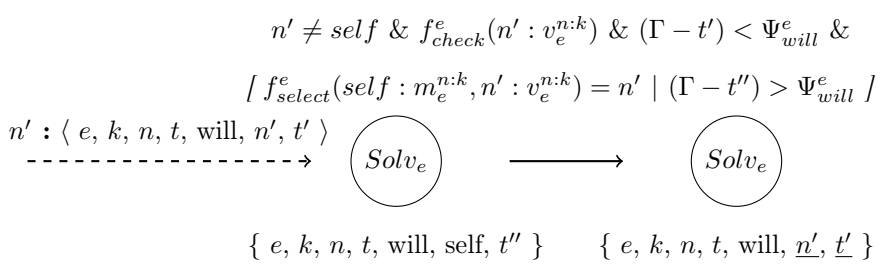
- la vue transmise par le solveur n' doit être validée par le filtre f^e_{check} (pour rappel, ce filtre permet d'effectuer des contrôles de sécurité sur les vues reçues) ;
- l'intervalle de temps écoulé depuis le verrouillage de la mission par n' (*i.e.* t') ne doit pas être supérieur à Ψ^e_{will} ;
- l'un des deux cas distincts doit se présenter :
  - l'intervalle de temps écoulé depuis le verrouillage de la mission par le solveur *self* (*i.e.* t'') est dépassé : le verrou du solveur *self* est donc périmé ;
  - le filtre f^e_{select} indique que le solveur n' est le plus pertinent pour terminer la mission $m^{n:k}_e$ concernée.

Les vues étant diffusées de façon régulière et la communication concernée par cette règle étant effectuée de manière directe, il y a une forte probabilité que les deux solveurs puissent exécuter la règle en parallèle. Dans le cas où un seul solveur reçoit la vue provenant du second solveur, l'élection locale peut conduire à deux situations distinctes :

- Le solveur ayant reçu la vue $v^{n:k}_e$ provenant du second solveur garde son verrou. Par conséquent, les deux solveurs continuent à travailler sur la même mission.

— Le solveur ayant reçu la vue $v_e^{n:k}$ relâche son verrou au profit du second solveur. Par conséquent, la mission $m_e^{n:k}$ est dorénavant traitée par un seul solveur.

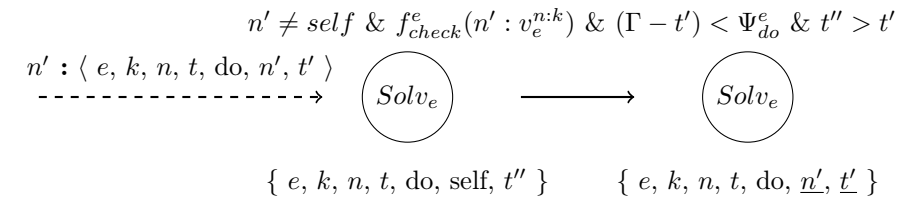
Dans ces deux situations, la mission $m_e^{n:k}$ ne se retrouve jamais orpheline (*i.e.* sans verrou). La condition de communication directe entre les solveurs est nécessaire au maintien de cette propriété. Le filtre f_{select}^e doit néanmoins avoir un comportement "cohérent" pour que cette propriété soit conservée. Par exemple, une sélection purement aléatoire pourrait en effet conduire une mission à devenir orpheline.



(9)

FIGURE 7.22 – Sélection locale entre deux solveurs pour une mission verrouillée en *will* en temps absolu.

La règle représentée en figure 7.23 permet à un solveur de relâcher son verrou en *do* sur une mission si un autre solveur ayant verrouillé la même mission lui communique sans intermédiaire cette information. Le fonctionnement ainsi que les conditions d'application de cette règle sont similaires à ceux de la règle précédente. En revanche, le filtre f_{select}^e ne prend pas ici part à la condition d'application de la règle. En effet, nous considérons que deux solveurs qui sont en train de traiter la même mission (*i.e.* ils ont un verrou en *do* sur la même mission) ont une pertinence similaire. Par conséquent, l'élection ne porte que sur les dates des verrous. Le solveur possédant le verrou le plus ancien est ici prioritaire dans la limite de la durée Ψ_{do}^e fixée.

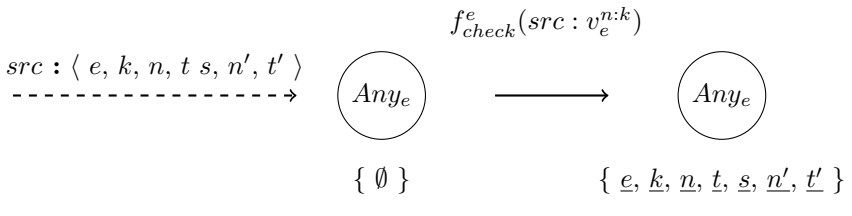


(10)

FIGURE 7.23 – Sélection locale entre deux solveurs pour une mission verrouillée en *do* en temps absolu.

Règles relatives à la réception d'une vue

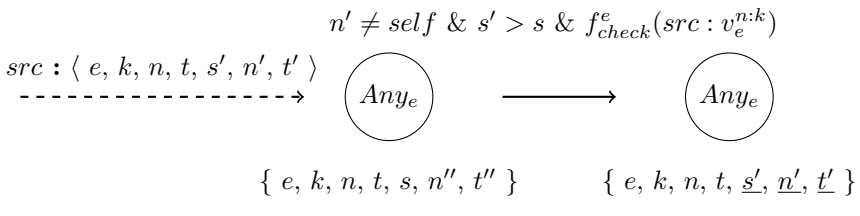
La règle représentée en figure 7.24 est celle qui permet d'intégrer une nouvelle mission provenant d'une vue communiquée par un autre nœud de la flotte. Les seules conditions préalables sont la validation de la vue par le filtre f_{check}^e et le fait que la mission ainsi intégrée ne soit pas déjà connue par le nœud appliquant la règle.



(11)

FIGURE 7.24 – Réception d'une vue d'une mission inconnue en temps absolu.

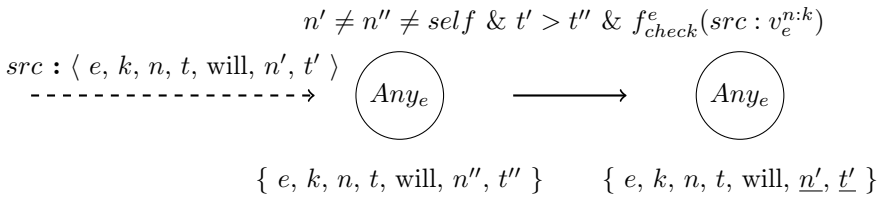
La règle représentée en figure 7.25 permet de mettre à jour une mission suite à la réception d'une vue de cette même mission contenant un état plus avancé.



(12)

FIGURE 7.25 – Réception d'une vue d'une mission en temps absolu.

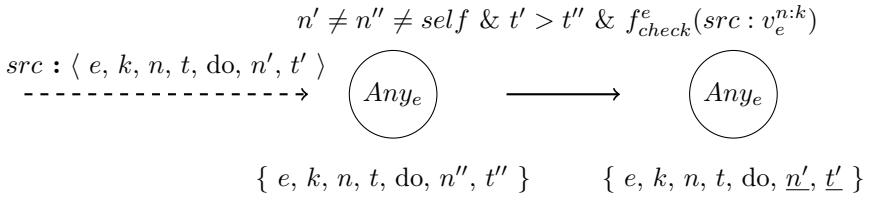
La règle représentée en figure 7.26 permet de mettre à jour une mission verrouillée en *will* suite à la réception d'une vue spécifiant que le propriétaire du verrou a changé. Cette règle n'est effectuée que si la date de dernière modification de la vue reçue est plus récente que celle de la mission stockée localement. Le verrou le plus récent est donc privilégié. Le nœud appliquant cette règle ne doit pas être impliqué dans le mission concernée afin de préserver un comportement cohérent avec avec la règle 9 (*c.f.* figure 7.22).



(13)

FIGURE 7.26 – Réception d'une vue d'une mission verrouillée en *will* en temps absolu.

La figure 7.27 est similaire à la précédente mais est dédiée aux missions verrouillées en *do*.



(14)

FIGURE 7.27 – Réception d'une vue d'une mission verrouillée en *do* en temps absolu.

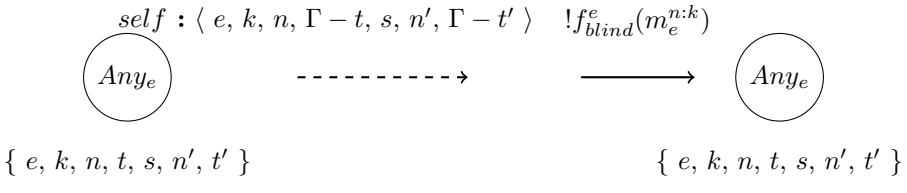
7.3.3 Règles du modèle en temps relatif

Dans l'approche par temps relatif, les horloges ne sont pas synchronisées de manière globale. Pour pallier à ce problème, les vues transportent des intervalles de temps (*i.e.* Δ) plutôt que des dates. Pour une mission $m_e^{n:k}$ représentée par le 7-uplet $\{e, k, n, t, s, n', t'\}$, la vue relative $v_e^{n:k}$ sera exprimée par $\langle e, k, n, \Delta, s, n', \Delta' \rangle$ où $\Delta = \Gamma - t$ et $\Delta' = \Gamma - t'$; Γ représentant la valeur actuelle de l'horloge interne du nœud émetteur de la vue $v_e^{n:k}$. Cette approche permet donc de corriger la différence de configuration initiale des horloges. En revanche, elle ne prend pas en compte le temps de transmission des vues entre les nœuds ni les faibles différences de vitesse (*i.e.* la dérive) qui peuvent exister entre ces horloges. L'estimation de l'âge des verrous peut donc différer légèrement d'un nœud à l'autre.

Les règles de ce modèle sont similaires à celles de l'approche par temps absolu à la différence que leurs conditions sont adaptées pour prendre en compte l'utilisation d'intervalles de temps à la place des dates.

Règle relative à l'envoi d'une vue

La règle 7.28 est une adaptation de la règle 7.21 qui utilise des intervalles de temps plutôt que des dates dans la vue diffusée. Les informations temporelles contenues dans chaque mission étant stockées sous forme de dates, les intervalles de temps sont calculés par rapport à la valeur de l'horloge locale du nœud appliquant la règle.

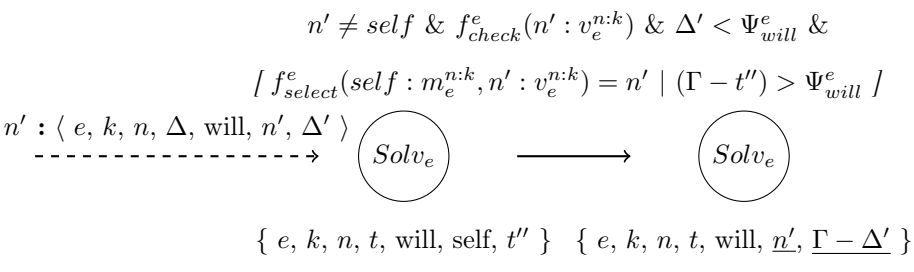


(8)

FIGURE 7.28 – Envoi d'une vue en temps relatif.

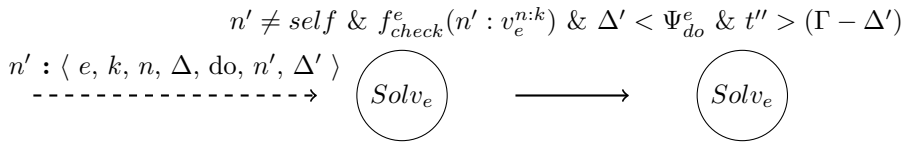
Règles relatives à la réception d'une vue par un solveur

De façon similaire, la règle 7.29 inclut un recalage des intervalles de temps contenus dans la vue reçue afin de les convertir en références temporelles absolues par rapport à la valeur de l'horloge locale du nœud appliquant la règle.



(9)

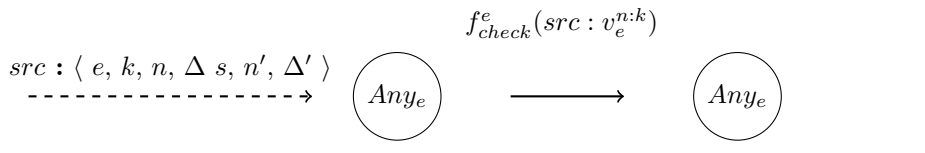
FIGURE 7.29 – Sélection locale entre deux solveurs pour une mission verrouillée en *will* en temps relatif.



(10)

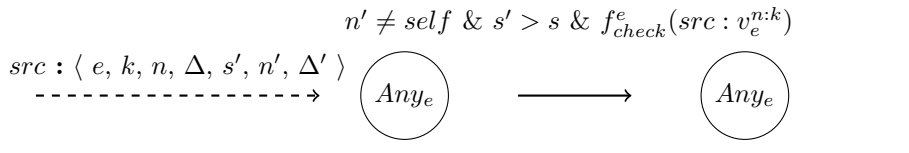
FIGURE 7.30 – Sélection locale entre deux solveurs pour une mission verrouillée en *do* en temps relatif.

Règles relatives à la réception d'une vue



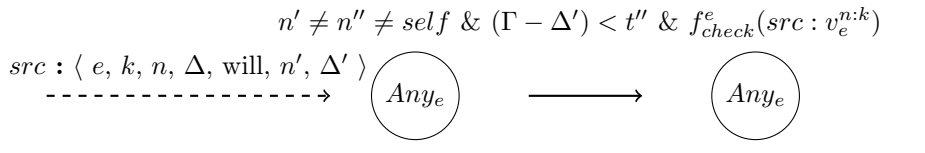
(11)

FIGURE 7.31 – Réception d'une vue d'une mission inconnue en temps relatif.



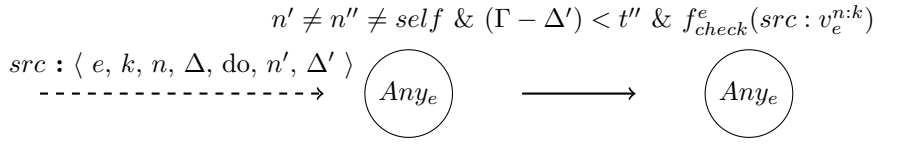
(12)

FIGURE 7.32 – Réception d'une vue d'une mission en temps relatif.



(13)

FIGURE 7.33 – Réception d'une vue d'une mission verrouillée en *will* en temps relatif.



(14)

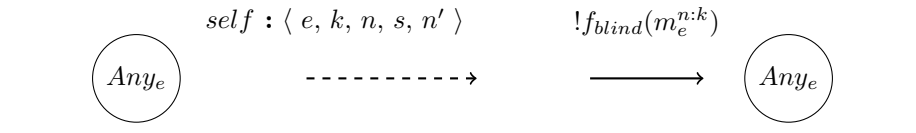
FIGURE 7.34 – Réception d'une vue d'une mission verrouillée en *do* en temps relatif.

7.3.4 Règles du modèle en ordre relatif

Tout comme le modèle par temps relatif, l'approche par ordre relatif ne requiert aucune synchronisation d'horloge. Dans cette approche, aucune information temporelle n'est partagée entre les nœuds de la flotte. Ainsi, la vue $v_e^{n:k}$ de la mission $m_e^{n:k}$ est exprimée par le 5-uplet $\langle e, k, n, s, n' \rangle$.

Les mesures de durées pour les verrous sont calculées par rapport à la date d'intégration de la mission dans l'ensemble de missions du nœud récepteur et non par rapport à une date donnée. Par conséquent, l'estimation de l'âge des verrous diffère d'un nœud à l'autre. Cette stratégie implique une fréquence de mise à jour des missions moins importante pouvant ainsi conduire à une augmentation des durées de leur traitement.

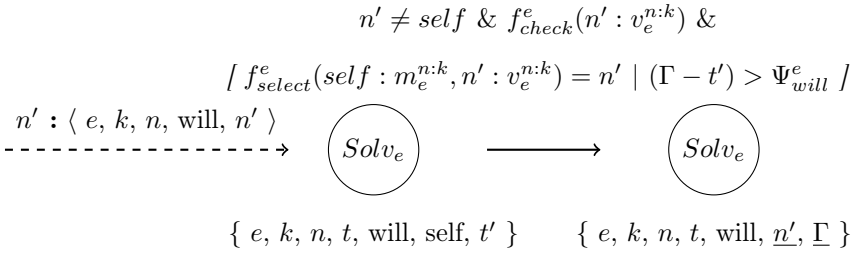
Règle relative à l'envoi d'une vue



(8)

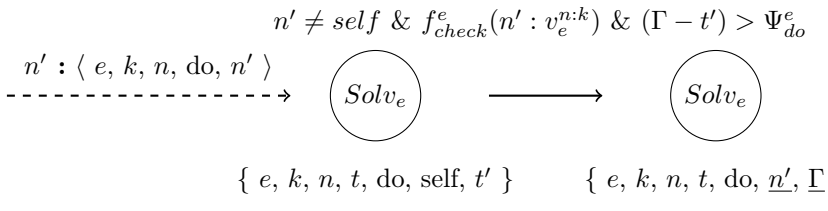
FIGURE 7.35 – Envoi d'une vue en ordre relatif.

Règles relatives à la réception d'une vue par un solveur



(9)

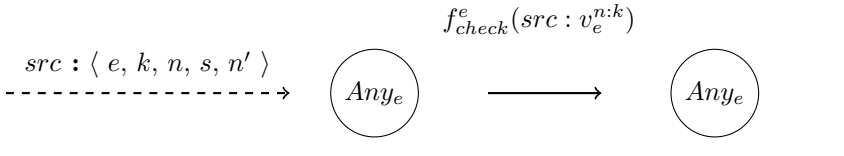
FIGURE 7.36 – Sélection locale entre deux solveurs pour une mission verrouillée en *will* en ordre relatif.



(10)

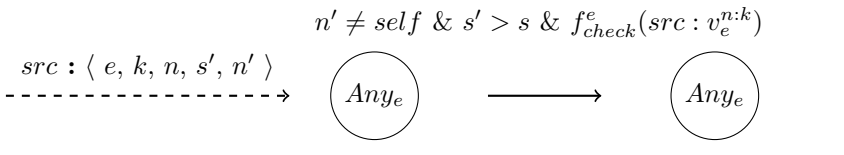
FIGURE 7.37 – Sélection locale entre deux solveurs pour une mission verrouillée en *do* en ordre relatif.

Règles relatives à la réception d'une vue



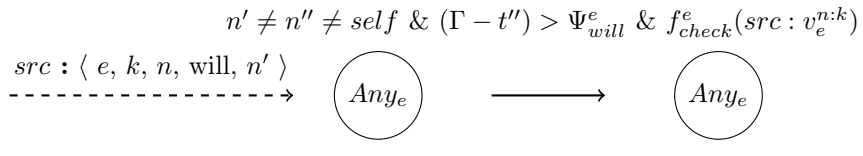
(11) $\{\ \emptyset\ \}$ $\{\ \underline{e}, \underline{k}, \underline{n}, \underline{\Gamma}, \underline{s}, \underline{n'}, \underline{\Gamma}\ \}$

FIGURE 7.38 – Réception d'une vue d'une mission inconnue en ordre relatif.



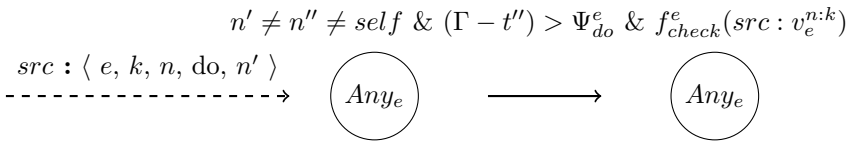
(12) $\{\ e, k, n, t, s, n'', t''\ \}$ $\{\ e, k, n, t, \underline{s'}, \underline{n'}, \underline{\Gamma}\ \}$

FIGURE 7.39 – Réception d'une vue d'une mission en ordre relatif.



(13) $\{\ e, k, n, t,$ will, $n'', t''\ \}$ $\{\ e, k, n, t,$ will, $\underline{n'}, \underline{\Gamma}\ \}$

FIGURE 7.40 – Réception d'une vue d'une mission verrouillée en *will* en ordre relatif.



(14) $\{\ e, k, n, t,$ do, $n'', t''\ \}$ $\{\ e, k, n, t,$ do, $\underline{n'}, \underline{\Gamma}\ \}$

FIGURE 7.41 – Réception d'une vue d'une mission verrouillée en *do* en ordre relatif.

7.3.5 Règles exceptionnelles

Dans cette section, nous présentons deux règles particulières permettant à l'application d'imposer à AMiRALE une modification d'une mission donnée. Ces règles ont pour objectif de lever certaines limitations du modèle qui peuvent, dans certains cas, brider l'application. Leur usage doit néanmoins rester exceptionnel au risque de perturber la capacité collaborative du système. La première règle, appelée *règle de préemption asynchrone*, permet à un solveur de posséder un verrou sur plus d'une mission au même moment. Elle lui permet également de prendre arbitrairement la main sur une mission déjà verrouillé. La seconde, appelée *règle d'annulation*, permet à un capteur d'annuler une mission en cours de traitement.

Règle de préemption asynchrone

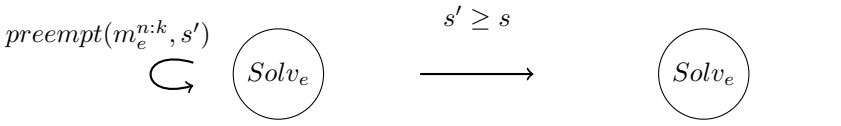
Dans le chapitre 6, nous avons énoncé deux principes concernant le verrouillage des missions :

— une mission ne doit être verrouillée que par un seul solveur à un instant donné ;

— un solveur ne peut verrouiller qu'une seule mission à un instant donné.

Les règles que nous avons présentées dans ce chapitre tentent de maintenir l'ensemble des missions dans un état qui respecte ces deux principes. Néanmoins, il existe des scénarios où de telles restrictions peuvent brider le comportement de certaines entités de la flotte. Prenons, par exemple, le cas d'un scénario de collecte dans lequel des entités doivent ramasser un ensemble d'objets présents au sol. Une mission est créé lorsqu'un nouvel objet à ramasser est détecté ; la donnée de la mission est ici la position de cet objet. Supposons qu'un solveur ayant verrouillé une mission se dirige vers l'objet concerné. Sur son chemin, imaginons qu'il trouve un autre objet relatif à une autre mission et qu'il est capable de traiter. Les contraintes actuelles des règles que nous avons proposées ne lui permettent pas de ramasser cet objet et donc de résoudre la mission correspondante.

C'est pourquoi nous introduisons le mécanisme de préemption asynchrone qui permet à une application de mettre explicitement à jour une mission en levant le principe d'unicité du verrou pour un même solveur. Cette règle, représentée à la figure 7.42, n'intervient que si elle est directement invoquée par l'application.



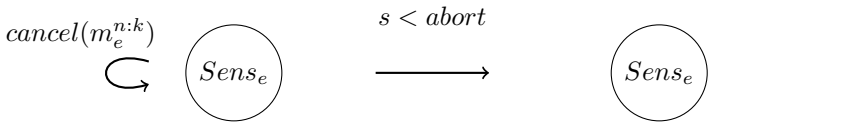
(15)

FIGURE 7.42 – Préemption asynchrone d'une mission par un solveur.

Règle d'annulation

Dans certains cas, il est possible qu'un capteur détermine lui même qu'une mission est invalide. Néanmoins, aucune des règles précédemment proposées ne lui permet d'intégrer cette information. En effet, un capteur n'est autorisé à modifier les informations d'une mission que lors de sa création.

La figure 7.43 présente une règle permettant à un capteur d'annuler (*i.e.* de passer en état *abort*) une mission donnée si l'application lui indique que cette dernière est invalide.



(16)

FIGURE 7.43 – Annulation d'une mission par un capteur.

7.4 Automates finis décrivant l'évolution de l'état d'une mission

Dans cette section nous présentons l'évolution de l'état d'une mission pour chaque rôle associé (capteur, solveur et relais). Nous illustrons cette évolution par un automate fini où les états de l'automate représentent les valeurs possibles de l'état s (*i.e. start*, *will*, *do*, *abort* et *end*) d'une mission. Les transitions correspondent à l'application des différentes règles qui peuvent intervenir dans la modification de l'état courant d'une mission. Les états $will\_self$ et do_self indiquent que la mission est verrouillée respectivement en *will* ou en *do* par l'entité concernée par l'automate.

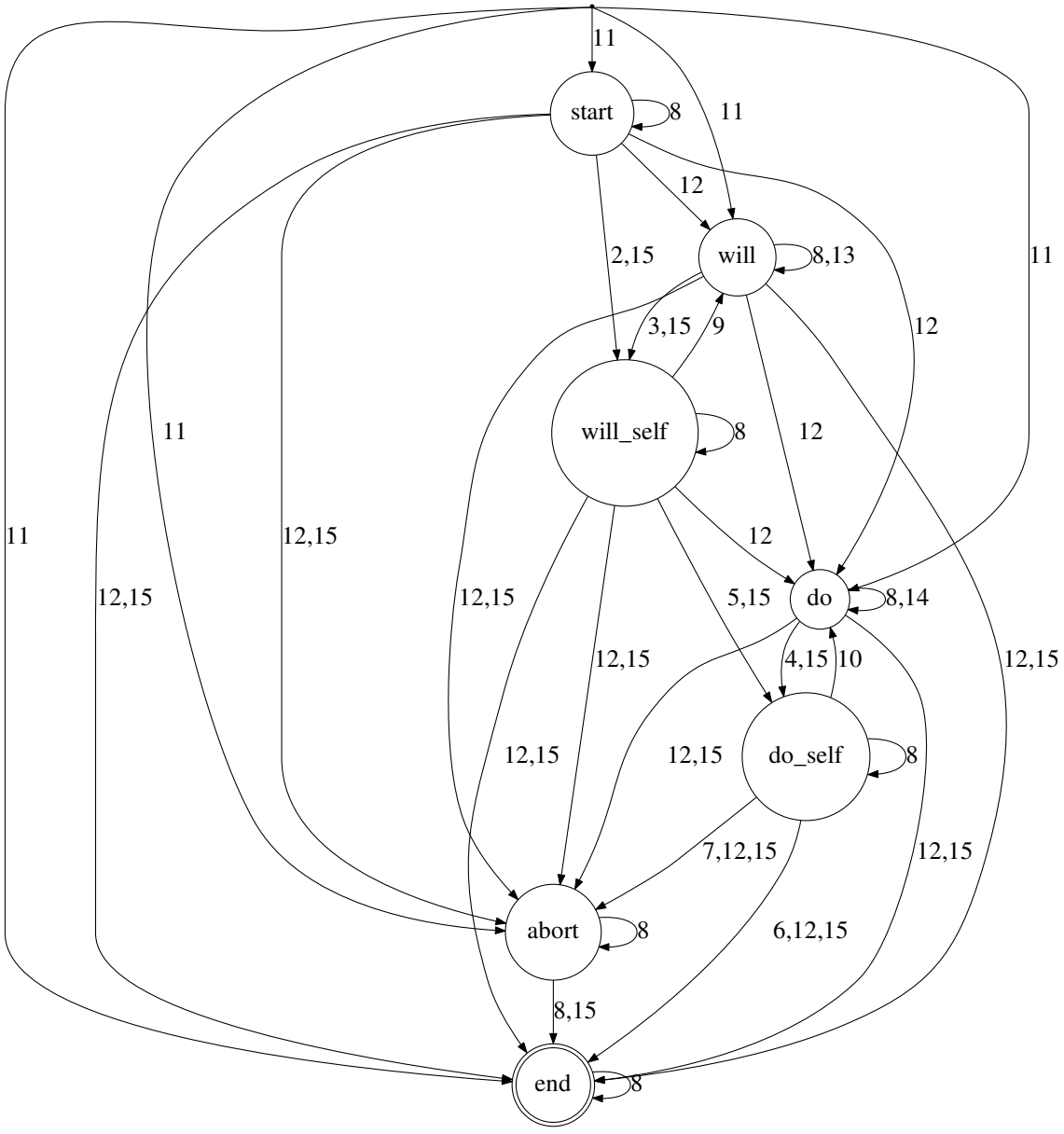


FIGURE 7.44 – Automate fini représentant l'évolution de l'état d'une mission pour un solveur.

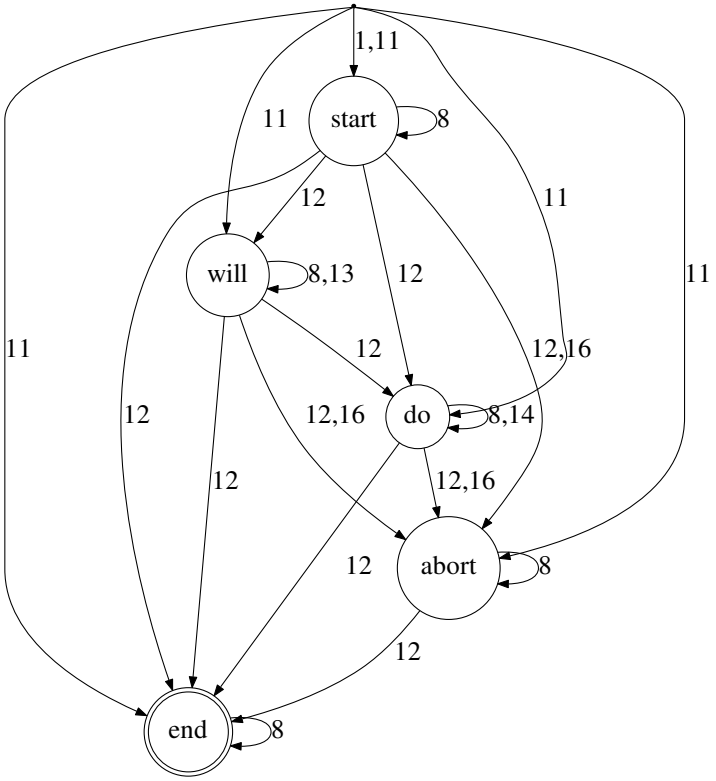


FIGURE 7.45 – Automate fini représentant l'évolution de l'état d'une mission pour un capteur.

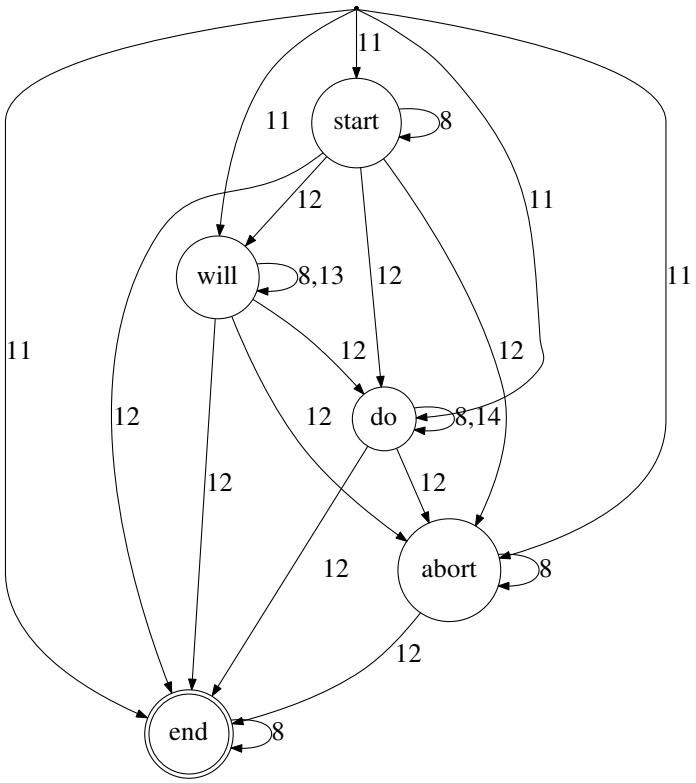


FIGURE 7.46 – Automate fini représentant l'évolution de l'état d'une mission pour un relais.

7.5 Résumé

Dans ce chapitre, nous avons présenté les différents modèles de ré-étiquetage de graphes sur lesquels nous nous sommes basés pour formaliser AMiRALE. Après avoir défini le formalisme général utilisé, nous avons présenté l'ensemble des règles de notre système collaboratif. Nous avons également détaillé les différentes règles spécifiques pour chacune des approches de synchronisation temporelle présentées au chapitre 6. Nous avons enfin présenté deux règles dites exceptionnelles permettant, sous certaines conditions, de permettre à l'application de lever certaines limitations de notre modèle.

III Expérimentations

CHAPITRE 8

Plate-forme virtuelle pour les réseaux mobiles

Sommaire

8.1 Motivations

L'expérimentation est un point angulaire dans le cycle de développement d'une application, permettant de tester, évaluer et approuver son utilisation ainsi que son degré de maturité. L'expérimentation sur des algorithmes ou des protocoles peut être faite par *simulation*. Cette technique est réalisée grâce à des simulateurs comme *ns* [232], *OMNeT++* [233] ou encore *JbotSim*[1] [234, 235], et permet d'apprécier le bon fonctionnement, l'efficacité ainsi que le passage à l'échelle de modèles d'algorithmes ou de protocoles.

L'expérimentation sur des implémentations diffère dans la mesure où elle opère sur des applications natives et s'intéresse davantage au temps d'exécution, à l'utilisation du processeur et de la mémoire, au trafic réseau généré ainsi qu'aux différentes réactions du système d'exploitation. Il est difficile d'effectuer cette tâche quand l'application repose sur une infrastructure impliquant plusieurs dizaines de machines. De plus, la mobilité et la dynamique de la topologie requises par certaines expérimentations complexifient la mise en œuvre de cette tâche. En effet, utiliser directement

1. Cet outil sera décrit dans le prochain chapitre.

Internet comme plate-forme de test est difficilement praticable dans la mesure ou peu de paramètres peuvent être contrôlés. Acquérir sa propre infrastructure nécessite d'importants moyens, tant sur le plan financier que spatial. De plus, la dynamique ou encore la mobilité de la topologie d'une telle plate-forme sont fortement restreintes. La virtualisation permet à la fois de réduire les coûts, mais également de simplifier les manipulations. Par ailleurs, il est prouvé que la virtualisation permet de réduire les dépenses énergétiques [236].

Un réseau virtuel utilise des machines virtuelles, au lieu d'utiliser directement des machines physiques, et les inter-connecte par des liens virtuels dans le but de générer une topologie arbitraire. Ces entités virtuelles peuvent être hébergées sur un ou plusieurs hôtes physiques afin d'outrepasser toute limite matérielle due à la taille de la topologie. La topologie peut évoluer au cours du temps que ce soit manuellement, *i.e.* par l'action directe d'un utilisateur, ou bien automatiquement en suivant un scénario de mobilité préalablement défini.

Dans ce chapitre, nous proposons un outil appelé NEmu permettant de générer, configurer et administrer des réseaux virtuels afin de tester et de valider des prototypes d'applications pour réseaux statiques, dynamiques ou mobiles avec un contrôle accru sur les propriétés de la topologie, sur la mobilité ainsi que sur la configuration des nœuds et des inter-connexions. Le but de cet outil est donc de permettre la création d'un réseau virtuel de taille raisonnable en minimisant l'équipement matériel nécessaire à sa réalisation. Le nom NEmu, pour ***N****etwork* ***E****mulator for* ***m****obile* ***u****niverses*, fait à la fois référence à sa capacité à générer des réseaux émulés (et non simulés) mais aussi à l'émulateur QEMU [237] qui est utilisé au sein de NEmu.

NEmu sera utilisé dans le prochain chapitre comme plate-forme virtuelle afin de réaliser des expérimentations en milieu mobile sur un prototype de notre système collaboratif AMiRALE présenté dans la partie II de ce manuscrit.

8.2 Description de NEmu

8.2.1 L'émulateur réseau

Description générale

NEmu est un programme d'environ 8000 lignes, écrit en python, permettant de créer un réseau distribué de machines virtuelles. Il est basé sur un concept appelé NVE [238] (*Network Virtualization Environment*). La principale caractéristique d'un NVE est de pouvoir abriter plusieurs réseaux virtuels (*Virtual Network* - VN) qui sont indépendants les uns des autres. Par défaut, un VN n'est pas conscient de l'existence d'un autre VN en cas d'hébergement partagé sur une même infrastructure physique. Un VN est un ensemble de *nœuds virtuels* inter-connectés par des *liens virtuels* formant une topologie réseau émulée. NEmu permet de construire plusieurs VNs en assurant une séparation stricte de chaque réseau afin de garantir l'intégrité de chacune des topologies. NEmu intègre l'ensemble des propriétés fondamentales qui définissent un NVE :

— La *flexibilité* et l'*hétérogénéité* permettent à l'utilisateur de construire une topologie arbitraire et hétérogène constituée de nœuds et de liens virtuels eux-mêmes paramétrables.
— L'*extensibilité* (ou *passage à l'échelle*) permet aux différents nœuds d'une même topologie d'être hébergés sur plusieurs hôtes physiques afin d'outrepasser les limitations d'une seule machine physique.
— L'*isolation* garantie une stricte séparation entre chaque VN qui vit au sein de la même infrastructure physique, et également entre les VNs et l'infrastructure elle même.
— La *stabilité* assure que des erreurs se produisant dans un VN ne peuvent affecter un autre VN.
— La *maintenabilité* permet à un VN d'être complètement indépendant de l'infrastructure physique qui l'héberge. Ainsi, ce même VN pourra être re-déployé au sein d'une autre infrastructure.
— Le *support hérité* permet au NVE d'émuler des composants anciens.
— La *programmabilité* fournit des services réseaux auxiliaires facilitant l'utilisation du VN (comme DHCP ou DNS par exemple).

De plus, NEmu inclut quatre propriétés supplémentaires :

— L'*accessibilité* permet à NEmu d'être exécuté sans aucun droit d'accès particulier sur l'infrastructure physique. En effet, la majeure partie des instances publiques, telles que les universités ou les laboratoires, ne fournissent pas de droits supérieurs aux utilisateurs sur leur infrastructure afin de prévenir tout risque de sécurité ou d'intégrité sur l'ensemble du domaine. C'est pourquoi NEmu fonctionne en espace utilisateur.
— La *dynamique* de la topologie permet aux nœuds de joindre ou de quitter un VN à tout moment sans perturber celui-ci. Un lien virtuel peut également être instancié ou supprimé à tout moment et à tout endroit dans un VN.
— La *mobilité* des nœuds permet de créer une topologie évolutive dans le temps et l'espace. En d'autres termes, la mobilité permet de créer des topologies dont les inter-connexions varient de manière autonome au cours du temps suivant un schéma donné.
— L'*aspect communautaire* permet à plusieurs utilisateurs d'inter-connecter différents VNs dans le but de créer un réseau plus large. Dans ce cas, chaque VN du réseau communautaire peut être considéré comme un système autonome (au sens réseau du terme). Un ensemble d'utilisateurs pourra également former un unique AS composé de plusieurs VNs.

L'ensemble des propriétés présentées ci-dessus forme une version augmentée du concept NVE que nous définissons comme NVE+. NEmu peut être considéré comme une *Infrastructure as a Service* virtuelle [239] (*virtual IaaS*) ; *i.e.* une infrastructure où l'utilisateur est libre de fixer l'ensemble des différents paramètres topologiques, tant au niveau des nœuds que des liens d'inter-connexion.

Éléments réseaux

NEmu est un générateur de réseaux virtuels distribués qui permet à un ou plusieurs utilisateurs de créer une topologie arbitraire et dynamique. À cet effet, NEmu est basé sur plusieurs briques distinctes. En effet, NEmu utilise des *nœuds virtuels* inter-connectés par des *liens virtuels* dans le but de créer une topologie réseau virtuelle. Cette topologie peut être hébergée sur une unique machine physique ou bien sur plusieurs hôtes afin de permettre d'étendre la taille du réseau virtuel. La partie du réseau reposant sur un même hôte représente une *session* qui est configurée au travers du *manager* NEmu.

Les nœuds virtuels

Un *nœud virtuel* dans NEmu (appelé `VNode`) est une machine virtuelle émulée, par l'intermédiaire de l'émulateur QEMU [237] qui requiert une unité de stockage principale hébergeant son système d'exploitation. Cette unité est généralement représentée par une image disque présente sur l'hôte physique. Deux types distincts de *nœuds virtuels* existent au sein de NEmu :

- Un `VHost` est une *machine virtuelle utilisateur* entièrement configurable (*e.g.* composants internes, périphériques, système d'exploitation).
- Un `VRouter` est un *routeur virtuel* directement configuré par NEmu proposant différents services "clés en main" dans le but de simplifier la gestion du réseau émulé.

Chaque nœud virtuel utilise une ou plusieurs *unités de stockage* qui peuvent être représentées par un périphérique physique (*e.g.* disque dur, disque externe) ou bien par un système virtuel tel que :

- Une *image native* qui stocke l'ensemble d'une image disque dans un seul fichier. Ce dernier peut être obtenu en utilisant divers formats directement fournis par QEMU (*e.g. raw, vdi, vmdk, vpc*). Cette image peut être formatée selon le système de fichiers souhaité (*e.g. ext, fat, ntfs, reiserfs, btrfs*).
- Un *fichier à trou*, appelé aussi *sparse file* ou *CoW* (*Copy on Write*) qui est le clone d'une image disque mais qui ne stocke que les différences avec celle-ci. Un fichier CoW est donc très léger en comparaison du fichier original. Ce type de fichier est notamment utilisé lorsque plusieurs `VNode` souhaitent utiliser la même image native.
- Un système de fichier *squash* qui est une image disque en lecture seule générée à partir d'un répertoire présent sur l'hôte physique.
- Une émulation du système de fichiers FAT16 ou bien une interface *virtio* [240] sur un répertoire de l'hôte physique qui permettent de rendre celui-ci accessible en lecture/écriture à une machine virtuelle.
- Une image disque accessible à une ou plusieurs machines virtuelles au travers du réseau IP en utilisant un protocole parmi NDB [241], SSH, HTTP, HTTPS, FTP, FTPT ou encore TFTP. Ce mécanisme permet de différencier l'hôte hébergeant une image disque de celui qui héberge les machines virtuelles qui utilisent cette image.

Un `VHost` nécessite une image disque créée au préalable et fournie par l'utilisateur. Contrairement à un `VHost`, un `VRouter` est directement configuré par NEmu. Son unité de stockage est générée de manière interne par NEmu et contient un ensemble de services facilitant la gestion du réseau virtuel. Ainsi les `VRouter` proposent différents serveurs (*e.g.* DHCP, DNS, NFS, HTTP, SSH, NTP), des protocoles de routage dynamique (*e.g.* RIP, OSPF), un pare-feu Netfilter [242] ainsi qu'une couche QoS avec *Traffic Control* [243]. Il est possible d'inclure de nouveaux services en utilisant un mécanisme de *plugins* fourni par NEmu. Un `VRouter` repose sur une version modifiée de *TinyCore* qui est une distribution Linux légère (environ 30 Mo) et hautement configurable [244]. Un `VRouter` requiert environ 100 Mo en mémoire vive en activant la totalité de ses services.

Les liens virtuels

Un *lien virtuel* dans NEmu (appelé `VLink`) est une inter-connexion réseau émulée entre deux entités virtuelles. Cette inter-connexion peut être faite directement au sein de l'émulateur du nœud (dans ces conditions, le lien sera attaché au nœud) ou bien réalisée par un programme tiers. Trois types de liens virtuels existent au sein de NEmu :

— Un `VLine` est un *lien virtuel point-à-point* qui inter-connecte deux entités virtuelles. On peut donc considérer ce lien comme un câble réseau.
— Un `VHub` est un *lien virtuel multi-points* émulant un concentrateur Ethernet (*hub*) et pouvant inter-connecter un ensemble d'entités virtuelles.
— Un `VSwitch` est un *lien virtuel multi-points* émulant un commutateur Ethernet (*switch*) et pouvant inter-connecter un ensemble d'entités virtuelles.

Les *liens virtuels* transportent des trames Ethernet d'une interface réseau virtuelle à une ou plusieurs autres interfaces. Le trafic Ethernet est transporté entre les interfaces réseaux par des tunnels mis en place par des connexions TCP ou UDP. NEmu peut également utilisé un VDE [245] (*Virtual Distributed Ethernet*) qui est un composant virtuel tiers permettant d'inter-connecter des `VNode` au travers d'un mécanisme de mémoire partagée fourni par le noyau Linux. Ce composant permet ainsi d'émuler les trois types de `VLink` présentés précédemment.

Nous avons conçu un programme nommé *vnd* [246] (*Virtual Network Device*), composé de 5000 lignes de C++, pouvant à la fois émuler un `VLine`, un `VHub` ou bien un `VSwitch`. L'avantage de ce programme réside dans son faible poids en mémoire (environ 250 Ko) ainsi que dans ses paramètres réseaux hautement configurables. En effet, il est possible au sein d'un *vnd* de régler à chaud la bande passante, le délai, la gigue ainsi que le taux d'erreurs sur n'importe quelle de ses interfaces et ce de manière strictement indépendante. Ceci nous permet d'obtenir un réglage plus fin de la topologie réseau. De plus, il est possible de spécifier la longueur de la file d'attente des paquets mais également d'associer un VLAN à chaque port du *vnd*. Chaque flux de données (*i.e.* connexion en entrée ou sortie sur un port) peut être configuré de manière indépendante.

En outre, NEmu intègre le type de lien `Slirp` fourni par QEMU. Ce denier est un lien spécial qui permet à une machine virtuelle d'accéder au réseau réel. Ce système utilise une technique proche du NAT afin de permettre à une machine virtuelle d'utiliser les cartes réseaux physiques de l'hôte qui l'héberge. Enfin, NEmu permet

de connecter des interfaces réseaux virtuelles en utilisant des tunnels TUN/TAP ou bien des sockets UNIX fournis par le noyau Linux. Le trafic d'une interface peut être sauvegardé dans un fichier *pcap* en vue d'être analysé par la suite [247].

La Figure 8.1 est un exemple de topologie réseau générée par NEmu. Sur le coté gauche, deux `VHost` sont inter-connectés à un `VRouter` au travers d'un `VSwitch` en utilisant des tunnels TCP. Sur le coté droit, deux autres `VHost` sont inter-connectés au même `VRouter` au travers d'un `VHub` en utilisant également des tunnels TCP. Ici, l'ensemble des liens virtuels sont gérés par des *vnd*.

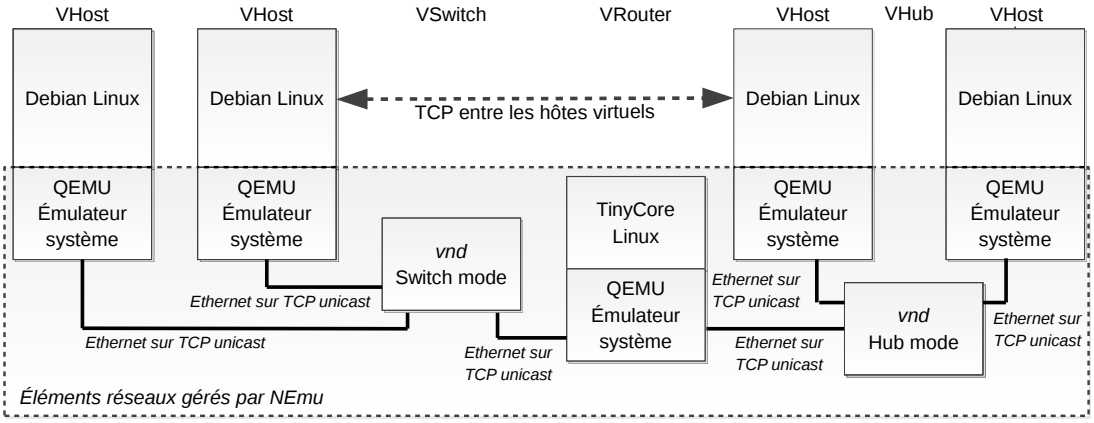


FIGURE 8.1 – Éléments réseaux de NEmu en action.

Les liens virtuels sans fil

Le composant *vnd* permet également d'émuler des `VLink` supplémentaires dédiés aux communications sans fil :

- `VAirAp` est un *lien virtuel multi-points* émulant à la fois un commutateur Ethernet mais également un point d'accès sans fil en mode *infrastructure*.
- `VAirWic` émule une carte réseau sans fil (*Wireless Interface Card* - WIC), pouvant opérer en mode *infrastructure* ou bien en mode *ad-hoc*. Le `VAirWic` est relié à une carte réseau virtuel filaire (*Network Interface Card* - NIC) de la machine virtuelle souhaitant l'utiliser. QEMU n'implémentant aucune carte réseau sans fil, le `VAirWic` fournit un tel équipement en le déportant sur un équipement virtuel dédié (*i.e.* le *vnd*).

Le trafic réseau sans fil est transporté par des tunnels mis en place par des connexions UDP.

Gestion du réseau virtuel

Les sessions

Une *session* au sein de NEmu représente la configuration d'une topologie réseau résidant sur un même hôte physique et appartenant à un unique utilisateur. Les *sessions* sont indépendantes même si elles font partie d'une unique topologie. Cela revient à dire qu'une erreur dans une *session* n'aura pas d'impact sur les autres. Un réseau virtuel distribué sur n machines physiques se composera au minimum de n *sessions*. De la même manière, un réseau virtuel fourni par n utilisateurs distincts

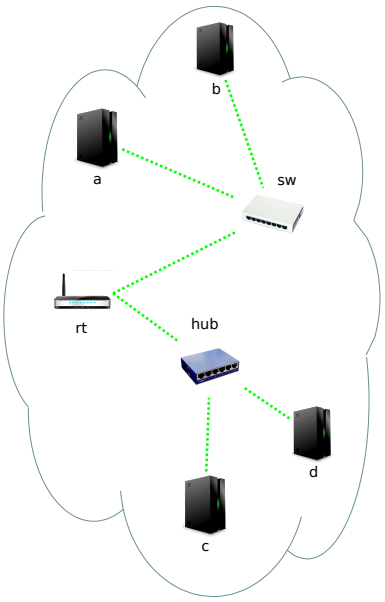
se composera au minimum de n *sessions*. Les éléments (*i.e.* `VNode` et `VLink`) de deux sessions distinctes peuvent néanmoins être reliés par l'intermédiaire de liens virtuels spéciaux appelés `VRemote` (`VAirRemote` dans le cas des liens virtuels sans fil). NEmu intègre une fonctionnalité de *migration* qui permet à un `VNode` d'être transféré à chaud d'une session à une autre. Les deux sessions concernées ne doivent pas nécessairement se trouver sur le même hôte physique. Une *session* est représentée par un système de fichiers auto-généré qui peut être sauvegardé et ré-utilisé par la suite sur la même (ou sur une autre) infrastructure. La figure 8.2 contient une représentation de la la topologie présentée en figure 8.1 ainsi qu'un exemple de script NEmu permettant de l'obtenir. Certains services (*e.g.* DHCP, DNS, SSH) sont activés sur le routeur présent dans cette topologie. La figure 8.3 utilise deux sessions distantes (*i.e.* ne s'exécutant pas sur le même hôte physique) pour former cette même topologie virtuelle.

Le manager

Le *manager* est le moyen qui permet à un utilisateur de manipuler une *session*. Il peut s'utiliser de trois façons distinctes :

— comme un module python classique à importer dans un code source ;
— comme un interpréteur de commandes dynamique ;
— comme un lanceur de scripts python.

Le *manager* permet également de manipuler plusieurs *sessions* au travers de la même interface en utilisant des tunnels SSH pour communiquer avec les *sessions* distantes. La topologie d'une session peut être visualisée à tout moment au travers de la suite logiciel *Graphviz* [248].



(a) *Illustration*

```
InitNemu()

VSwitch('sw', niface=3)
VHub('hub', niface=3)

VRouter("rt", nics=[VNic(), VNic()], services=[
 Service("ipforward"),
 Service("ifup", "0:192.168.1.1", "1:192.168.2.1"),
 Service("dnsmasq", domain="net1", net="192.168.1.0/24",
          start="192.168.1.10", end="192.168.1.20"),
 Service("dnsmasq", domain="net2", net="192.168.2.0/24",
          start="192.168.2.10", end="192.168.2.20"),
 Service("sshd")])

VHost('a', hds=[VFs('debian.img', type='cow')]
         , nics=[VNic()], m=512, cpu="coreduo")
VHost('b', hds=[VFs('debian.img', type='cow')]
         , nics=[VNic()], m=1024, cpu="core2duo")
VHost('c', hds=[VFs('debian.img', type='cow')]
         , nics=[VNic()], m=256, cpu="n270")
VHost('d', hds=[VFs('debian.img', type='cow')]
         , nics=[VNic()], m=2048, cpu="phenom")

Link('a', 'sw:1')
Link('b', 'sw:2')
Link('c', 'hub:1')
Link('d', 'hub:2')
Link('rt:0', 'sw:0')
Link('rt:1', 'hub:0')

StartNemu()
```

(b) *Script NEmu*

FIGURE 8.2 – Illustration et script NEmu de la topologie présentée en figure 8.1.

```
# Session 1 [10.0.0.1]

InitNemu()

VSwitch('sw', niface=3)

VRouter("rt", nics=[VNic(), VNic()], ...)

VHost('a', hds=[VFs('debian.img', type='cow')]
       , nics=[VNic()], ...)
VHost('b', hds=[VFs('debian.img', type='cow')]
       , nics=[VNic()], ...)

SetIface('rt:1', addr='10.0.0.2', port=8080)

Link('a', 'sw:1')
Link('b', 'sw:2')
Link('rt:0', 'sw:0')

StartNemu()
```

(a) *Script NEmu de la session 1*

```
# Session 2 [10.0.0.2]

InitNemu()

VHub('hub', niface=3)

VRemote('remote')

VHost('c', hds=[VFs('debian.img', type='cow')]
       , nics=[VNic()], ...)
VHost('d', hds=[VFs('debian.img', type='cow')]
       , nics=[VNic()], ...)

SetIface('remote', addr='10.0.0.1', port=8080)

Link('c', 'hub:1')
Link('d', 'hub:2')
Link('hub:0', 'remote'))

StartNemu()
```

(b) *Script NEmu de la session 2*

FIGURE 8.3 – Scripts NEmu distribués de la topologie présentée en figure 8.1.

8.2.2 Le composant d'inter-connexion

Description générale

Cette section propose une description détaillée du composant *vnd* utilisé au sein de NEmu afin d'émuler les différentes déclinaisons des liens virtuels `VLink` (*i.e.* câble, commutateur, concentrateur, point d'accès, et carte sans fil). Bien que ce ne soit pas le seul programme existant permettant d'émuler certains de ces équipements, le *vnd* dispose de plusieurs avantages et fonctionnalités qui se révèlent utiles dans le domaine de la virtualisation de réseaux :

- c'est un programme léger indépendant des machines virtuelles (*i.e.* une erreur du *vnd* ne porte pas atteinte à l'intégrité des machines virtuelles qui l'utilisent) ;
- il supporte les connexions et les déconnexions à chaud, ce qui lui permet d'être immunisé contre les fautes pouvant survenir dans les machines virtuelles qui l'utilisent ;
- il dispose de plusieurs mécanismes de transport dont les sockets réseaux classiques, lui conférant une grande portabilité et le rendant ainsi compatible avec des émulateurs autres que QEMU ;
- il permet de configurer dynamiquement et asymétriquement les propriétés des liens (*i.e.* débit, délai, gigue et taux d'erreurs) ;
- il fournit une émulation de liens et de composants sans fil aussi bien en mode *infrastructure* qu'en mode *ad-hoc* ;
- il ne nécessite aucun droit particulier sur l'infrastructure physique, lui conférant ainsi l'accessibilité inhérente au concept NVE+.

Architecture

Le *vnd* est composé d'un moteur et de plusieurs interfaces réseaux (*i.e.* ports) qui peuvent être créées, supprimées et configurées à chaud. La limitation du nombre de ces interfaces est uniquement liée à la puissance de la machine physique sur

laquelle le *vnd* s'exécute. Chaque interface possède ses propres files d'attente de paquets (*i.e.* file d'entrée et file de sortie) dont les tailles (*i.e.* nombre de tampons) peuvent être également configurées à chaud. Ces interfaces réseaux sont connectées au travers du moteur du *vnd*. La figure 8.4 illustre l'architecture d'un *vnd* possédant deux interfaces. Une donnée entrant dans un *vnd* peut être traitée de deux manières distinctes :

— *raw* : la donnée n'est pas interprétée. Elle est donc considérée comme un flux binaire. Chaque tampon (*buffer*) de la file d'attente peut contenir autant d'octets que sa configuration lui permet.

— *Ethernet* : la donnée est considérée comme une trame Ethernet. Dans ce cas, chaque tampon de la file d'attente ne peut contenir qu'une seule de ces trames. Ce comportement est notamment utilisé pour établir des communications entre les machines virtuelles émulées par QEMU.

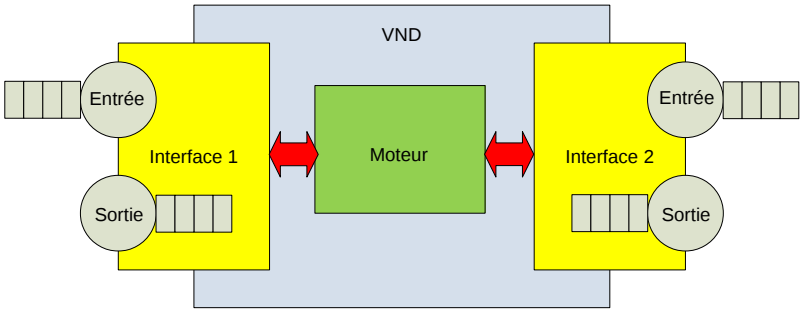


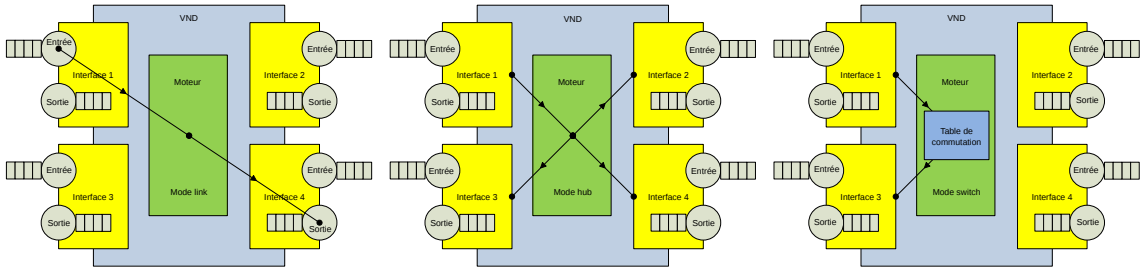
FIGURE 8.4 – Architecture d'un *vnd* possédant deux interfaces réseaux.

Le *vnd* supporte six modes de fonctionnement différents qui dépendent du composant réseau émulé. Les deux derniers modes sont utilisés afin d'émuler des cartes réseaux sans fil en mode *infrastructure* ou bien en mode *ad-hoc*. Dans ces deux cas, le *vnd* est utilisé comme composant réseau indépendant devant s'interfacer avec une machine virtuelle dans le but de lui conférer des capacités de communication sans fil. Le lien entre une machine virtuelle et un *vnd* émulant une WIC est fait par l'intermédiaire d'une interface réseau filaire. Les six modes de fonctionnement du *vnd* sont les suivants :

1. Dans le mode *link*, chaque interface est directement reliée à une autre interface, ce qui veut dire que toute donnée arrivant dans la file d'entrée d'une de ces interfaces est immédiatement transmise dans la file de sortie de la seconde interface. Une illustration de ce mode est présentée en figure 8.5a.
2. Dans le mode *hub*, chaque interface est reliée à toutes les autres, ce qui veut dire que toute donnée arrivant dans la file d'entrée d'une des interfaces est transmise dans la file de sortie de chacune des autres interfaces. Une illustration de ce mode est présentée en figure 8.5b.
3. Dans le mode *switch*, toute trame arrivant dans la file d'entrée d'une des interfaces est transmise au moteur du *vnd* qui redirige la donnée dans la file de sortie de la bonne interface en utilisant pour cela une table de commutation. Cette table est construite grâce aux adresses MAC contenues dans les trames. Une illustration de ce mode est présentée en figure 8.5c.

4. Dans le mode *access point*, pour une donnée entrante le *vnd* agit comme un *switch* ou bien comme un *hub* en fonction de la nature (*i.e.* filaire ou sans fil) de l'interface d'entrée et de celle de sortie.
5. Dans le mode *infrastructure*, le *vnd* se comporte comme un *link* faisant ainsi l'intermédiaire entre l'interface filaire et l'interface sans fil.
6. Dans le mode *ad-hoc*, toute trame arrivant dans la file d'entrée de l'interface filaire est transmise dans la file de sortie de chacune des autres interfaces sans fil. Inversement, toute trame arrivant dans la file d'entrée d'une interface sans fil est transmise dans la file de sortie de l'interface filaire.

Les quatre derniers modes n'ont de sens que si les données sont interprétées comme des trames Ethernet contenant ainsi des adresses MAC que le moteur du *vnd* peut exploiter. Afin d'émuler le protocole IEEE 802.11, un pseudo-en-tête est ajouté à chaque trame provenant d'un point d'accès ou d'une interface réseau sans fil.



(a) *Mode link* (b) *Mode hub* (c) *Mode switch*

FIGURE 8.5 – Architectures des principaux modes de fonctionnement du *vnd*.

La table de commutation contenue dans le *vnd* adopte le même comportement que celles contenues dans les commutateurs physiques, *i.e.* elle dispose d'un mécanisme d'auto-apprentissage. En effet, lorsqu'une trame est reçue par une interface, le moteur du *vnd* vérifie si l'adresse MAC source de la trame est associée à cette même interface. Si ce n'est pas le cas, le moteur intègre cette association (*i.e.* interface $\leftrightarrow$ adresse MAC) à la table de commutation. Dès lors, quand une trame est reçue, le moteur recherche l'interface associée à l'adresse MAC de destination de la trame et transmet celle-ci sur la file de sortie de l'interface ainsi trouvée. Actuellement, les associations contenues dans la table de commutation ont une durée de vie illimitée. Aucune suppression n'est donc opérée hormis en cas d'incohérence (*i.e.* adresse MAC et interface ne se correspondent plus) due à des changements topologiques ou bien en cas de suppression manuelle. Le *vnd* supporte l'utilisation des VLAN au niveau 1 (*i.e.* VLAN par port) sur chacune de ses interfaces.

Implémentation

Dans le domaine de la virtualisation, le terme *backend réseau* est souvent utilisé pour désigner la partie logicielle d'une émulateur qui permet d'établir une connexion

entre ce dernier et un autre composant virtuel potentiellement distant (*i.e.* se trouvant sur un hôte physique distinct). Les *backends réseaux* dans le monde UNIX sont généralement implémentés au travers des interfaces TUN/TAP du noyau Linux, des VDE ou bien des sockets TCP, UDP ou UNIX. Le *vnd* est compatible avec l'ensemble de ces *backends réseaux* au travers d'objets internes que nous appelons *endpoints*. Afin d'établir une communication, un *endpoint* est attaché (*bind*) à une interface réseau du *vnd*. Le *endpoint* représente une configuration réseau (*i.e.* protocole, adresse IP, port, etc.) tandis que l'interface englobe les files d'entrée et de sortie faisant ainsi l'intermédiaire entre le moteur du *vnd* et les communications réseaux réelles. Ces mécanismes d'attachement et de détachement opèrent à chaud permettant ainsi aux connexions et à la topologie réseau d'être dynamique. Une illustration d'un attachement entre un *endpoint* et une interface dans le *vnd* est présentée en figure 8.6.

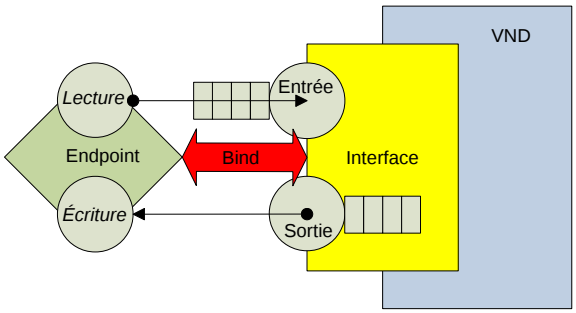


FIGURE 8.6 – Attachement d'un *endpoint* à une interface réseau dans un *vnd*.

8.2.3 Le mobilisateur

Description générale

NEmu est un émulateur de réseaux virtuels pouvant être mobiles. En d'autres termes, il est possible de créer un réseau virtuel dont la topologie est évolutive dans le temps de manière autonome. *nemo* [249] est un programme léger (environ 1 Mo en mémoire vive) écrit en C++, d'environ 3000 lignes, permettant de générer des scénarios de connectivité pour les réseaux mobiles. Un scénario de connectivité est une suite temporelle d'événements de connexion et de déconnexion entre des nœuds mobiles. Chaque connexion représente un lien sans fil. *nemo* est capable d'envoyer des ordres à NEmu en temps réel permettant ainsi d'émuler les changements de connectivité entre les nœuds mobiles en créant, détruisant ou changeant les caractéristiques des liens aux moments voulus. Tout comme le *vnd*, *nemo* peut être entièrement contrôlé par NEmu qui sert alors d'interface unique pour l'utilisateur. *nemo* se compose de deux parties : une partie basée sur un ordonnanceur en temps simulé et une autre basée sur un ordonnanceur en temps réel.

Ordonnanceur en temps simulé

L'ordonnanceur en temps simulé est le cœur de la partie simulation de *nemo*. Il permet de générer des scénarios de connectivité pour l'ordonnanceur en temps réel. Trois étapes sont nécessaires pour générer un scénario de connectivité :

1. Générer une *carte*, *i.e.* une zone géographique virtuelle dans laquelle les nœuds mobiles vont se déplacer.
2. Générer un *scénario de mobilité* sur cette carte. Un scénario de mobilité représente les informations relatives à la position et les mouvements de chaque nœud en fonction du temps (discrétisé). Un scénario de mobilité est donc une trace d'exécution d'un modèle de mobilité exécutée pendant un temps donné.
3. Générer un *scénario de connectivité* à partir du scénario de mobilité. Un scénario de connectivité représente l'ensemble des informations sur le changement d'état des liens (*i.e.* apparition, disparition, configuration) entre les nœuds en fonction du temps (discrétisé).

Après avoir généré le scénario de mobilité, l'ordonnanceur en temps simulé le déroule sur la carte fournie ; à chaque intervalle de temps (fixé par l'utilisateur), il calcule les distances et les éventuelles connexions sans fil possibles entre toutes les paires de nœuds mobiles. Son fonctionnement est illustré à la Figure 8.7. Les résultats de chaque étape sont sauvegardés pour être éventuellement rechargés ultérieurement afin d'éviter de les re-calculer. De plus, il est possible de fournir à *nemo* des scénarios de mobilité ou de connectivité externes, *i.e.* générés par un autre programme ou écrits à la main. *nemo* est capable d'importer des traces de mobilité provenant du simulateur *ns* [232] ou bien généré par l'outil de mobilité populaire *Bonnmotion* [250]. Nativement, *nemo* utilise le modèle de mobilité *Randow Waypoint* pour générer ses scénarios de mobilité.

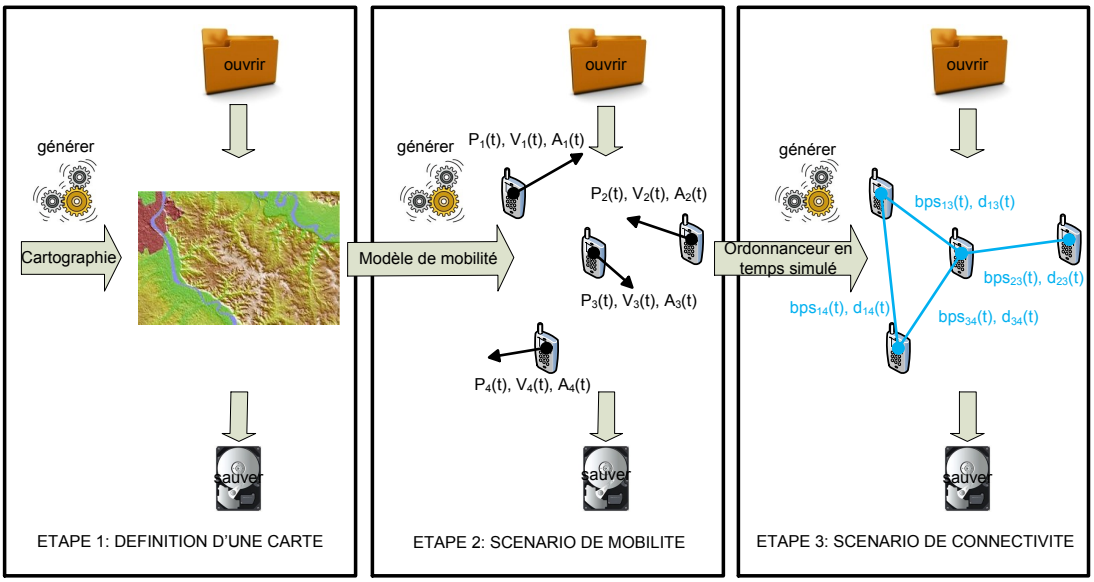


FIGURE 8.7 – Génération de scénarios de connectivité avec *nemo*.

L'ordonnanceur en temps simulé souffre de quelques limitations :
— il nécessite des calculs intensifs (*i.e.* complexité en $O(n^2)$ pour n nœuds) ;
— il y a un compromis à faire entre la précision temporelle (intervalle de temps entre chaque évaluation de la connectivité), la durée totale des calculs et le nombre d'événements détectés ;
— actuellement, les scénarios de connectivité ne peuvent être générés qu'en suivant le modèle de propagation *Two-ray Ground Reflection* et en considérant que les cartes sans fil suivent la norme IEEE 802.11G (Wi-Fi version *g*).

Ordonnanceur en temps réel

L'ordonnanceur en temps réel est le cœur de la partie émulation de *nemo*. Il exécute des événements de connectivité à leurs intervalles de temps exacts fixés par rapport au début du scénario. La précision temporelle utilisée dans l'ordonnanceur temps réel est égale à celle fixée lors du traitement du scénario de mobilité par l'ordonnanceur en temps simulé. Son fonctionnement est illustré à la Figure 8.8. Celle-ci montre que NEmu joue le rôle de contrôleur central vis à vis des autres composants. Dans un réseau virtuel mobile, les *vnd* font office de cartes réseaux sans fil pour les nœuds virtuels, c'est donc au sein des *vnd* que les communications réelles, par l'intermédiaire des *endpoints*, sont instanciées. NEmu récupère les événements de connectivité générés par *nemo* et les retransmet aux différents *vnd* correspondant aux cartes réseaux sans fil des différents nœuds virtuels afin de faire évoluer la topologie du réseau. L'ordonnanceur temps réel peut être mis en pause puis remis en marche à tout moment par l'utilisateur.

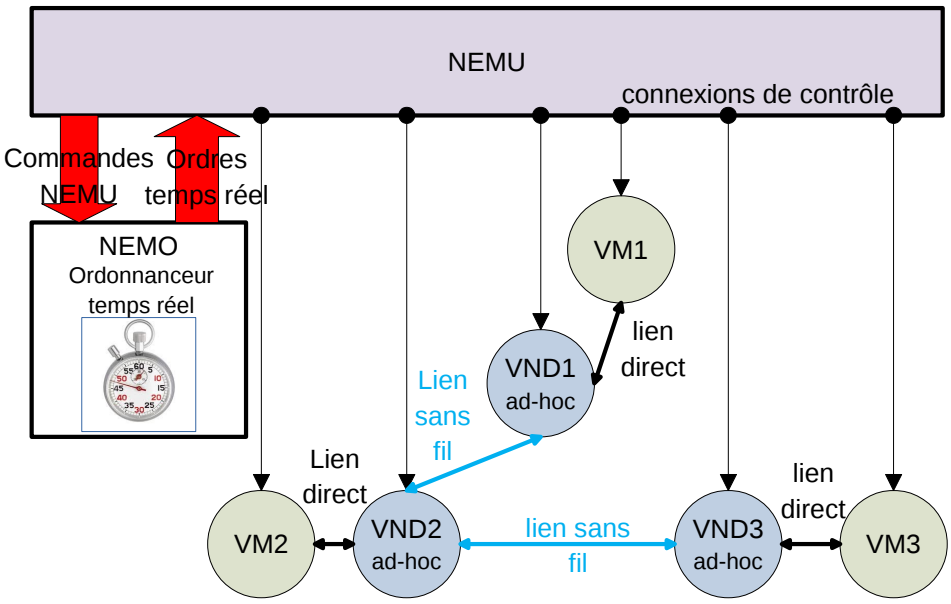


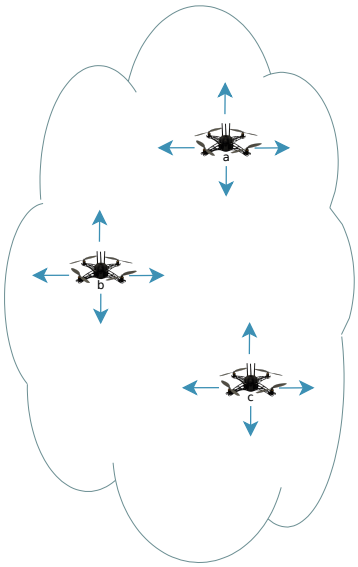
FIGURE 8.8 – Émulation de réseaux mobiles virtuels avec *nemo*.

L'ordonnanceur temps réel souffre de quelques limitations :

— contrairement à NEmu, *nemo* est centralisé ;

— les sockets réseaux utilisées pour inter-connecter les instances du *vnd* (i.e. les *cartes réseaux sans fil*) introduisent des délais et réduisent significativement la précision temporelle. Cette dernière sera au mieux de l'ordre de la milliseconde.

La figure 8.9 illustre l'utilisation du mobilisateur dans NEmu sur un scenario exploitant trois drones. Les inter-connexions entre les drones sont directement générées par *nemo* pour une durée de 120 secondes avec 100 événements de mobilité (*i.e.* changement de vitesse, d'accélération ou de direction pour un nœud). Les changements de connectivité à appliquer sont transmis à NEmu toutes les 5 secondes.



(a) *Illustration*

```
InitNemu()

VAirWic('awic')
VAirWic('bwic')
VAirWic('cwic')

VHost('a', hds=[VFs('drone.img', type='cow')]
         , nics=[VNic()], ...)
VHost('b', hds=[VFs('drone.img', type='cow')]
         , nics=[VNic()], ...)
VHost('c', hds=[VFs('drone.img', type='cow')]
         , nics=[VNic()], ...)

Link('a', 'awic')
Link('b', 'bwic')
Link('c', 'cwic')

MobNemu('mob', nodes=['awic', 'bwic', 'cwic'])

GenMobNemu('mob', width=1000, height=1000
                , time=120, events=100, step=5)

StartNemu()
StartMobNemu('mob')
```

(b) *Script NEmu*

FIGURE 8.9 – Illustration et script NEmu exploitant le mobilisateur *nemo*.

8.3 Expérimentation

Dans cette section, nous souhaitons mettre en évidence la validité des résultats des expérimentations conduites avec NEmu. Pour ce faire, nous reproduisons un test de performance initialement effectué sur *Mininet* [251].

Cet outil permet, tout comme NEmu, de créer des réseaux virtuels. Il est réputé dans le milieu académique pour le réalisme des résultats qu'il permet d'obtenir [252]. Néanmoins, Mininet ne permet pas nativement de créer des réseaux virtuels mobiles. De plus, Mininet repose principalement sur des fonctionnalités du noyau Linux qui nécessitent des droits particuliers sur l'infrastructure physique. En effet, il utilise des conteneurs *LXC* [253] afin de virtualiser les nœuds. Ce système permet d'encapsuler un groupe de processus ainsi qu'une partie du système de fichier dans un conteneur virtuel. Cette solution est donc plus légère que de l'émulation mais ne permet pas une

configuration aussi flexible que celle proposée par QEMU. Mininet utilise également l'outil *Open vSwitch* [254] afin d'établir des liens virtuels entre les conteneurs LXC. Cet outil permet, tout comme *vnd*, de configurer finement les propriétés des interconnexions. Néanmoins, il utilise les interfaces réseaux virtuelles du noyau Linux qui ne peuvent être créées et configurées que par un administrateur du système hôte. Enfin, *Open vSwitch* ne peut virtualiser aucun composant réseau sans-fil. De par tous ces aspects, Mininet n'est pas compatible avec le concept NVE+.

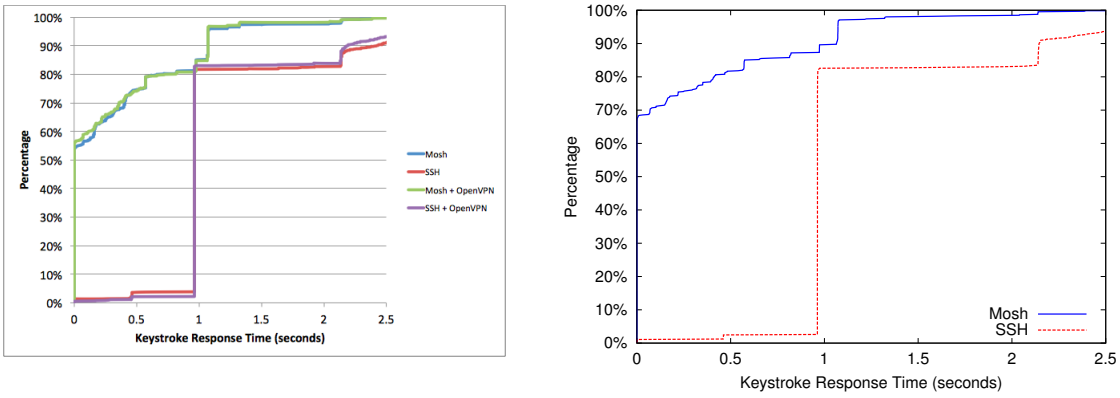
Notre objectif est de comparer les résultats d'une expérimentation d'un coté avec Mininet et de l'autre avec NEmu. L'expérimentation à laquelle nous nous intéressons concerne l'outil *Mosh* [255] (*Mobile Shell*). Mosh est une application permettant de contrôler un terminal à distance comme le fait SSH. Néanmoins, contrairement à ce dernier, Mosh repose sur le protocole SSP qui repose lui même sur UDP. Couplé à un algorithme de prédiction, Mosh est sensé délivrer une meilleur réactivité dans le cas où la connexion réseau est intermittente. L'expérimentation consiste à mesurer et à comparer le temps de réponse de frappes (*keystrokes response time*) de Mosh et de SSH. Les résultats originaux ont été obtenus en utilisant *Mininet* [251].

Nous avons reproduit une expérimentation effectuée à l'université de Standford [256] officiellement reprise par les développeurs de Mosh. Dans cette expérimentation, un serveur est connecté à un commutateur au travers d'un réseau 3G [99]. Le client est quant à lui relié au commutateur par l'intermédiaire d'une connexion Wi-Fi. Les auteurs considèrent les propriétés réseaux suivantes :

- Réseau 3G :
  - taux d'erreurs des paquets : 0.01 ;
  - bande passante : 1 Mbit/s ;
  - délai : 450 ms.
- Réseau Wi-Fi :
  - taux d'erreurs des paquets : 0.08 ;
  - bande passante : 25 Mbit/s ;
  - délai : 30 ms.

Grâce à notre composant *vnd*, nous avons configuré le réseau conformément aux spécifications ci-dessus.

Les résultats obtenus à l'université de Standford avec le simulateur Mininet sont présentés en figure 8.10a. Nos résultats obtenus avec notre émulateur NEmu sont présentés en figure 8.10b.



(a) *Résultats avec Mininet* (b) *Résultats avec NEmu*

FIGURE 8.10 – Résultats d'expérimentation sur Mosh et SSH avec Mininet et NEmu.

Nous pouvons remarquer que les résultats obtenus avec Mininet et ceux obtenus avec NEmu sont similaires. Les faibles différences de performance sont dues au fait que Mininet et NEmu ne reposent pas sur le même système de virtualisation. Ces résultats nous permettent de conclure que notre émulateur offre un degré de réalisme comparable à celui de Mininet.

8.4 Résumé

Dans ce chapitre, nous avons présenté un nouvel outil appelé NEmu permettant notamment de créer des réseaux virtuels mobiles et hautement configurables. Nous avons détaillé en section 8.1 l'ensemble des propriétés (*i.e.* le concept NVE+) qui ont permis d'aboutir à la création de notre outil. Après avoir décrit l'implémentation de NEmu (section 8.2), nous avons comparé les résultats d'une expérimentation obtenus d'une part sur NEmu, et d'autre part sur un outil d'expérimentations réseaux reconnu dans le milieu académique (section 8.3). Nous avons ainsi pu valider le réalisme des expérimentations pouvant être conduites sur notre outil.

CHAPITRE 9

Scénario de mise en œuvre et évaluation d'AMiRALE

Sommaire

9.1 Scénario ParCS

Afin d'évaluer les performances de notre système collaboratif AMiRALE, nous avons créé le scénario d'application ParCS (*Park Cleaning Swarm*) dont l'objectif est de permettre à une flotte autonome de robots terrestres (*i.e.* UGV) de collecter un ensemble des déchets présents dans un parc. Chaque robot est spécialisé dans le traitement d'un seul type de déchet (*e.g.* verre, plastique, papier, compost). Nous considérons que les déchets sont répartis de manière uniforme sur le sol du parc.

Nous distinguons deux déclinaisons de notre scénario de nettoyage :

Terrestre : chacun des robots est capable de détecter la présence de n'importe quel type de déchet. Il se déplace dans le parc suivant un modèle de mobilité

aléatoire afin de collecter les déchets compatibles (*i.e.* qu'il est capable de traiter).

Aérienne : les robots ne peuvent détecter que les déchets de leur propre type. En support des robots terrestres, des drones aériens (*i.e.* UAV) se déplaçant selon un modèle de mobilité réaliste (*c.f.* chapitre 3) sont utilisés pour détecter l'ensemble des déchets depuis les airs. L'altitude à laquelle ils évoluent leur confère une plus grande visibilité du sol et donc un rayon de détection plus important. Dans ce contexte, les robots terrestres ne sont mobiles que lorsqu'ils doivent traiter un déchet particulier repéré en amont par un drone. Ce dernier point a pour objectif d'économiser les batteries des robots terrestres.

Dans cette déclinaison aérienne, la flotte utilisée est donc hétérogène à deux niveaux :

1. les robots ne peuvent traiter (et détecter dans le cas du scénario aérien) qu'un seul type de déchet, ils disposent donc de capacités différentes ;
2. la déclinaison aérienne de notre scénario implique l'utilisation de deux types d'engins différents (*i.e.* drones et robots).

9.2 Objectifs et paramètres d'évaluation

Au travers du scénario ParCS, nous souhaitons évaluer notre système AMiRALE selon plusieurs paramètres :

- le nombre de déchets présents dans le parc ;
- le nombre de robots contenus dans la flotte ;
- le nombre de drones contenus dans la flotte ;
- le nombre de types de déchet ;
- le taux de panne des robots ;
- le modèle de mobilité utilisé ;
- certains paramètres de notre système AMiRALE :
  - la durée de validité des verrous Ψ_{do} et Ψ_{will} ;
  - la configuration du filtre f_{blind} permettant d'empêcher la diffusion d'une vue ;
  - l'usage de la règle de préemption asynchrone dans le traitement d'un déchet faisant l'objet d'une mission verrouillée.

En faisant varier ces paramètres, nous effectuons un certain nombre de mesures :

- le temps nécessaire au nettoyage de l'ensemble des déchets se trouvant dans le parc ;
- le nombre de missions en mémoire, ce qui nous permettra de déterminer la quantité de mémoire requise par notre système ;
- le nombre de vues par message, ce qui nous permettra de déterminer la quantité de bande passante requise par notre système ;
- la bande passante effective en entrée et en sortie consommée par AMiRALE ;
- la distance moyenne parcourue par un robot, la "quantité de déplacement" représentant une part importante de la consommation énergétique totale [257].

9.3 Utilisation du système AMiRALE

Dans le cadre de ce scénario, nous utilisons AMiRALE afin de permettre aux différents robots de la flotte de collaborer. Le type de mission e correspond au type de déchet. Un robot pouvant traiter un déchet de type e est donc un $Solv_e$. Tout $Solv_e$ est aussi $Sens_e$. Dans le cas de la déclinaison terrestre du scénario ParCS, chaque robot est $Sens_i$ pour tout type de mission i (*i.e.* tout robot de la flotte peut détecter n'importe quel déchet quelque-soit son type). Dans le cas de la déclinaison aérienne, seuls les drones sont $Sens_i$ pour tout type de mission i et un robot de la flotte ne peut détecter que les déchets de son propre type.

Lorsqu'un robot (ou un drone) détecte un déchet, il crée une nouvelle mission contenant la position de ce dernier. Chaque robot consulte régulièrement sa mémoire locale afin de vérifier si une mission compatible est disponible (*i.e.* mission non verrouillée ou disposant d'un verrou périmé). Si une telle mission existe, le robot la verrouille dans l'état *will* et se dirige vers l'emplacement du déchet. Lorsqu'il se trouve assez prêt du déchet pour pouvoir le collecter, il fait évoluer l'état de la mission en *do*. Lorsque le déchet est nettoyé, le robot fait passer la mission dans l'état final *end*. Si un déchet n'est pas trouvé à l'endroit indiqué par la mission, l'état de cette dernière est fixée à *abort* par le robot disposant du verrou.

Nous considérons que le nettoyage d'un déchet est une opération atomique. Un robot ne peut ramasser un déchet que si il possède un verrou sur une mission en rapport avec ce déchet. Tous les nœuds le flotte (*i.e.* drones et robots) diffusent l'ensemble de leurs vues toutes les 5 secondes (ce paramètre est choisi de façon arbitraire). Chaque message contient, en complément de la liste des vues, l'identifiant du nœud émetteur ainsi que sa position (nous considérons ici que les robots sont équipés d'un système de positionnement global).

Certaines optimisations sont faites par l'intermédiaire des filtres du modèle (*c.f.* section 6.3.2) :

— Lorsqu'un capteur détecte un déchet, il ne créé la mission relative que si sa mémoire ne contient pas de mission en cours relative à ce même déchet (filtre f^e_{ignore}).
— Un solveur sélectionne toujours la mission relative au déchet compatible le plus proche de sa position (filtre f^e_{pass}).
— Si deux solveurs traitent une même mission et que ces derniers sont assez proches pour communiquer directement, le solveur le plus éloigné géographiquement du déchet abandonne la mission au profit du solveur le plus proche (f^e_{select}).

9.4 Évaluations et résultats

9.4.1 Évaluation de la version par temps absolu

Principe général

Dans cette section, nous comparons les performances de notre système AMiRALE dans sa version par temps absolu avec deux autres modèles :

— le *pire* : une méthode collective ne reposant sur aucune communication entre les robots que nous appelons *Mute*.

— le *meilleur* : une méthode collaborative centralisée reposant sur une mémoire partagée (*Blackboard*).

Nous utilisons ici la déclinaison terrestre de notre scénario ParCS.

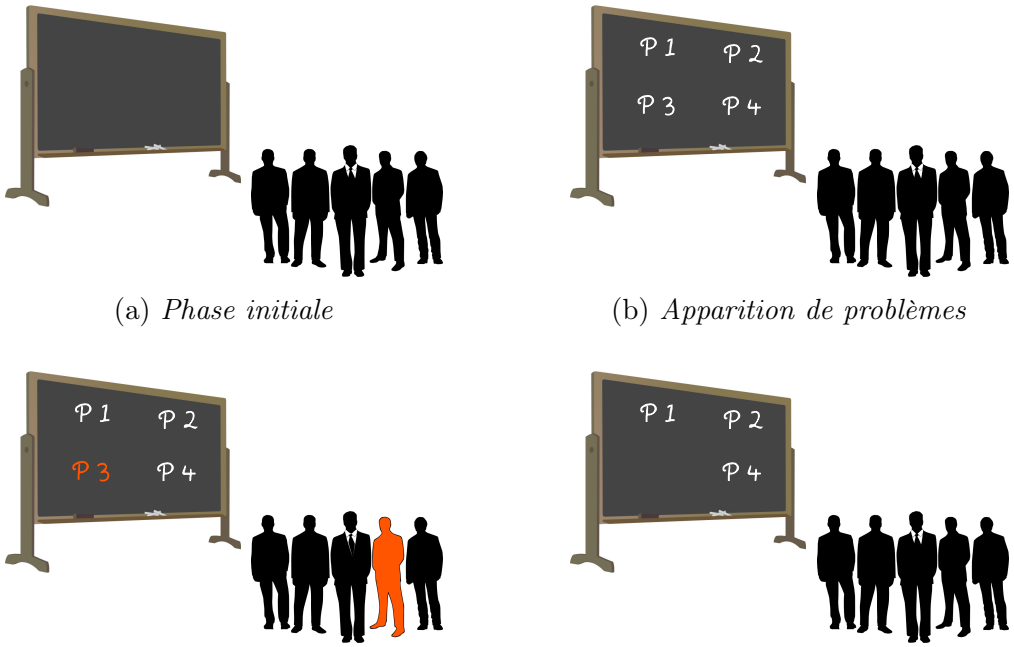
Modèle collectif *Mute*

Une méthode collective (*c.f.* section 1.3.2) implique que les robots ne peuvent pas communiquer les uns avec les autres. Dans cette solution, chaque robot se déplace de manière aléatoire, ramassant les déchets qu'il est capable de traiter lorsqu'il en est assez proche. Cette approche est la pire dans le sens où aucune collaboration n'a lieu entre les entités de la flotte. Elle est donc sensée fournir des résultats en dessous de ceux de notre système AMiRALE.

Modèle collaboratif *Blackboard*

La méthode collaborative centralisée à laquelle nous nous intéressons est le *Blackboard* à mémoire partagée que nous avons évoqué en section 1.3.4. Le principe général du Blackboard est illustré à la figure 9.1. Ce système permet à un ensemble d'entités spécialisées (*i.e.* experts) de résoudre un problème complexe. Ce dernier est subdivisé en tâches individuelles (*i.e.* problèmes) qui nécessitent des compétences particulières pour être résolues. La subdivision implique que chaque problème est réalisable par au moins un des experts. Ces problèmes sont répertoriés sur un tableau noir (*i.e.* le *blackboard*). Un expert souhaitant résoudre un problème particulier le verrouille sur le *blackboard* avant de commencer son travail, il indique ainsi aux autres experts que ce problème est en cours de résolution. Une fois résolu, le problème est supprimé du *blackboard* (ou bien marqué comme résolu). Dans le cadre de notre scénario de nettoyage de parc, les experts sont les robots de la flotte. Tout comme pour le modèle Mute, les robots se déplacent de manière aléatoire ramassant les déchets qu'ils sont aptes à traiter. Lorsqu'un robot détecte un déchet incompatible, la position de ce dernier est référencée sur le *blackboard*. Chaque robot consulte périodiquement le contenu du *blackboard* à la recherche du déchet compatible le plus proche. Lorsqu'un tel déchet est disponible, sa position référencée sur le *blackboard* est verrouillée par le robot qui se déplace alors directement vers celle-ci. Lorsque le déchet est supprimé, l'information correspondante est effacée du *blackboard*. La préemption n'est ici pas autorisée, ce qui veut dire qu'un déchet verrouillé ne peut être traité que par le robot qui a posé le verrou. Lorsqu'un robot tombe en panne, le déchet potentiellement verrouillé par ce dernier est immédiatement déverrouillé afin d'éviter tout risque d'inter-blocage. Le *blackboard* à mémoire partagée est sensé être

le meilleur modèle collaboratif bien qu'il soit complètement irréaliste [74]. En effet, tous les nœuds disposant d'une unique et même mémoire, tout événement, action ou connaissance d'un nœud est immédiatement disponible pour les autres. Ce partage ne nécessite aucune communication entre les nœuds de la flotte. Ce modèle est donc théoriquement le meilleur dans la mesure où la dissémination de l'information est atomique.



(a) *Phase initiale* (b) *Apparition de problèmes*

(c) *Verrouillage d'un problème* (d) *Suppression d'un problème*

FIGURE 9.1 – Illustration du fonctionnement du modèle *Blackboard*.

Configuration d'AMiRALE

En complément de la configuration présentée à la section 9.3, deux optimisations supplémentaires sont ajoutées à notre système AMiRALE :

— Un nœud empêche la diffusion d'une mission terminée depuis plus de 1000 secondes sans contradiction (filtre f_{blind}).

— La durée de validité d'un verrou est fixée à 1000 secondes (Ψ_{will} et Ψ_{do}).

Une vue reçue par un nœud provoque une contradiction si celle-ci contient des informations différentes (*i.e.* plus récentes ou plus anciennes) de celles appartenant à la mission relative stockée en mémoire du nœud. Le cas où le nœud ne contient aucune mission en mémoire relative à la vue reçue est également considéré comme une contradiction.

Paramètres de simulation

L'ensemble des résultats présentés dans cette section ont été obtenus à l'aide de l'outil *JbotSim* [234, 235]. Ce simulateur de graphes dynamiques écrit en JAVA

permet d'exécuter et de visualiser des algorithmes distribués exploitant des objets mobiles hétérogènes et communicants.

Pour les trois solutions (Mute, Blackboard et AMiRALE), les robots se déplacent suivant le modèle de mobilité *Random Waypoint* (*c.f.* section 3.2.1). Chaque point sur chaque courbe représente une valeur moyenne obtenue sur 500 simulations. Nos paramètres de simulation sont décrits dans le tableau 9.1.

TABLE 9.1 – Paramètres de simulation de la déclinaison terrestre du scénario ParCS.

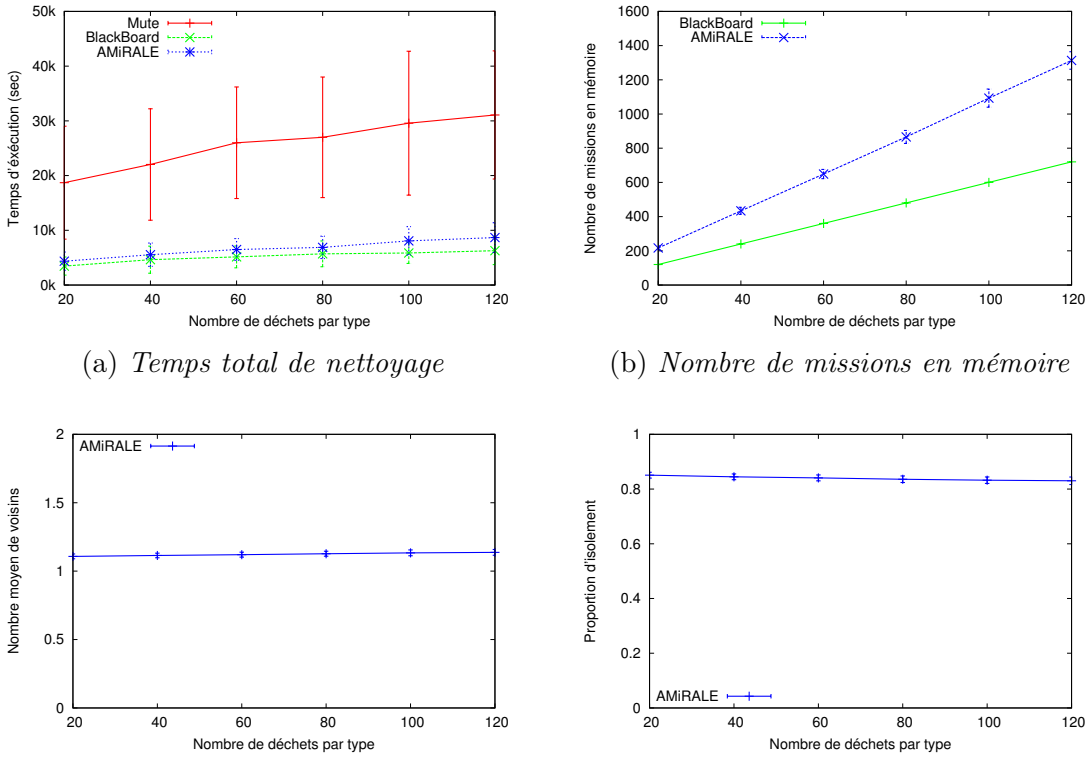
Paramètre	**Valeur**
Simulateur	JBotSim
Taille du parc	1000 m x 1000 m
Taille des robots	1 m x 1 m
Vitesse des robots	5 m / s
Modèle de mobilité	*Random Waypoint*
Fréquence des communications	5 s
Rayon de détection des robots	30 m
Rayon de communication des robots	30 m
Nombre d'exécutions par point	500

Nous évaluons tout d'abord le temps d'exécution nécessaire au nettoyage de la totalité du parc en fonction de plusieurs paramètres tels que le nombre de types de déchet, le nombre de déchets par type, le nombre de robots par type ainsi que la fréquence de pannes des robots. Dans nos simulations, une panne se traduit par la suppression d'un robot (et donc la perte de sa mémoire locale) et par son remplacement immédiat par un robot de même type (à mémoire initiale vide). Nous relevons également la différence du nombre de missions en mémoire entre le Blackboard et AMiRALE en fonction des mêmes paramètres. Nous mesurons enfin la connectivité moyenne par robot en fonction de ces mêmes paramètres. La connectivité moyenne est évaluée par le nombre moyen de voisins par nœud sur l'ensemble de la simulation ainsi que la proportion moyenne de temps durant laquelle les nœuds sont complètement isolés. Notre dernier résultat est une fonction de répartition cumulative du temps de ramassage des déchets présents dans le parc.

Résultats de simulation

La figure 9.2 présente nos résultats de simulation en fonction du nombre de déchets par type. La figure 9.2a représente le temps total de nettoyage du parc pour les différentes solutions (Mute, Blackboard et AMiRALE). La figure 9.2b montre le nombre moyen de missions en mémoire pour chaque robot en fin scénario pour les solutions Blackboard et AMiRALE (le modèle Mute ne possède pas de mémoire). Enfin les figures 9.2c et 9.2d représentent respectivement le nombre moyen de voisins ainsi que le taux d'isolement moyen par robot. Ces deux derniers résultats ont uniquement été mesurés pour notre système AMiRALE. Les modèles Mute et Blackboard n'utilisant pas de communication entre les robots, la connectivité des nœuds leur est indifférente. Le nombre de types de déchet ainsi que le nombre de robots

par type sont fixés à 6. Le nombre de robots ainsi que le nombre de déchets sont donnés par type ; *i.e.* pour 6 robots par type, il y a donc 36 robots au total.



(a) *Temps total de nettoyage* (b) *Nombre de missions en mémoire*

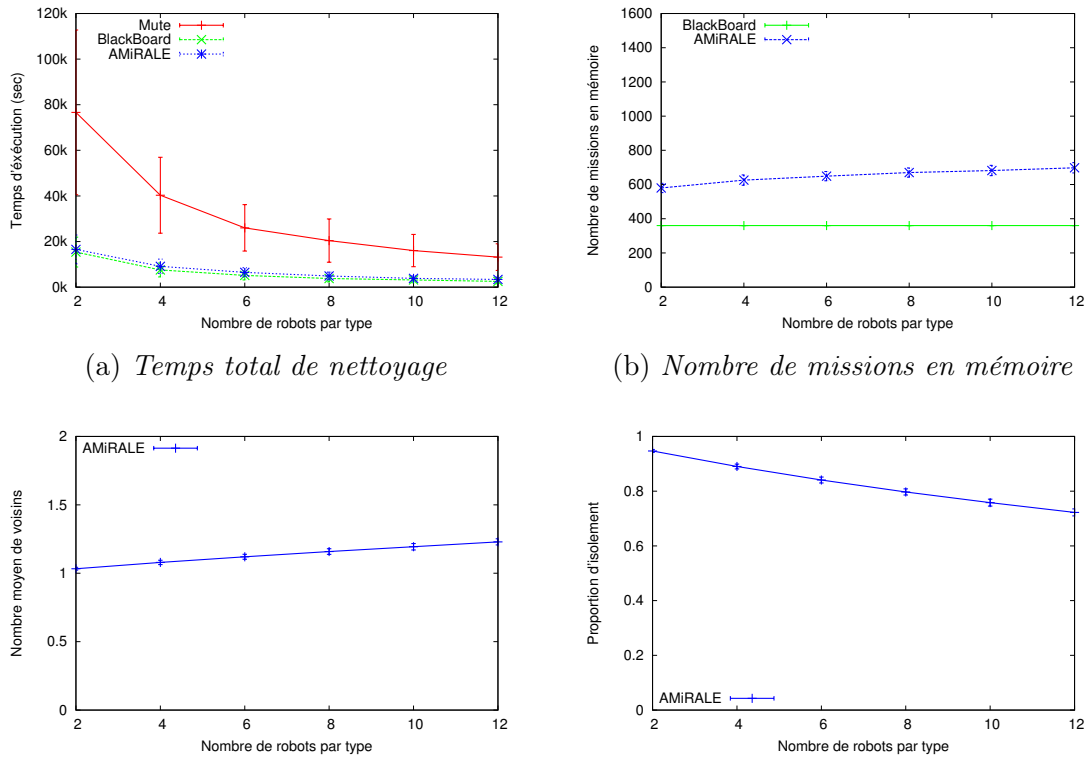
(c) *Nombre de voisins par robot* (d) *Proportion d'isolement par robot*

FIGURE 9.2 – Résultats de simulation pour AMiRALE en temps absolu en fonction du nombre de déchets/type.

Comme nous pouvions le prévoir, le temps d'exécution augmente lorsque le nombre de déchets présents dans le parc est plus important. Les valeurs importantes des écart-types de la courbe du modèle Mute sont dues à l'aspect purement aléatoire de ce modèle. En effet, chaque robot se déplace aléatoirement et indépendamment des autres, ramassant les déchets compatibles lors de son passage. Le temps d'exécution est fortement couplé à la position initial des robots, à la distribution des déchets dans le parc ainsi qu'aux changements de direction induits par le modèle de mobilité. Ces raisons conduisent à des temps d'exécution très fluctuants pour le modèle Mute. Nous pouvons également remarquer que la différence de temps de nettoyage entre le modèle Blackboard et AMiRALE est faible. Le Blackboard reste néanmoins plus efficace dans la mesure où aucune communication n'est nécessaire pour disséminer les informations sur les déchets découverts ; ce modèle ne souffre donc pas des effets de distance et de partage d'informations. En effet, reposant sur une mémoire partagée, aucune information relative au même déchet ne peut différer entre les robots qui utilisent le Blackboard. Le nombre moyen de missions contenues dans la mémoire de chaque robot en fin de scénario augmente linéairement avec le nombre de déchets présents dans le parc. Le nombre de déchets est rigoureusement égal au nombre de missions générées dans le cas du modèle Blackboard. AMiRALE génère quant à lui environ deux fois plus de missions qu'il n'existe de déchets. La

raison est que deux capteurs peuvent initier une mission différente pour un même déchet malgré la présence du filtre f_{ignore}. Ce filtre empêche la création d'une mission uniquement si le capteur a connaissance de l'existence d'une mission équivalente. Or, en raison de l'aspect distribué de notre modèle, un nœud ne possède qu'une vision locale de la connaissance globale de la flotte. C'est la raison pour laquelle des missions supplémentaires sont présentes en mémoire. Nos courbes présentant le nombre moyen de voisins par nœud ainsi que la proportion de temps pour laquelle les nœuds sont isolés corroborent ces résultats. Nous pouvons en effet remarquer que la flotte est fortement déconnectée, les nœuds étant isolés plus de 80 % du temps total du scénario. Enfin, nous soulignons le fait que la connectivité de la flotte n'est que très peu impactée par la variation du nombre de déchets présents dans le parc.

La figure 9.3 s'intéresse aux mêmes indicateurs (*i.e.* temps total de nettoyage, nombre de missions en mémoire, nombre moyen de voisins et proportion d'isolement) en fonction du nombre de robots par type. Le nombre de types de déchet est fixé à 6, tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 360 déchets au total).



(a) *Temps total de nettoyage* (b) *Nombre de missions en mémoire*

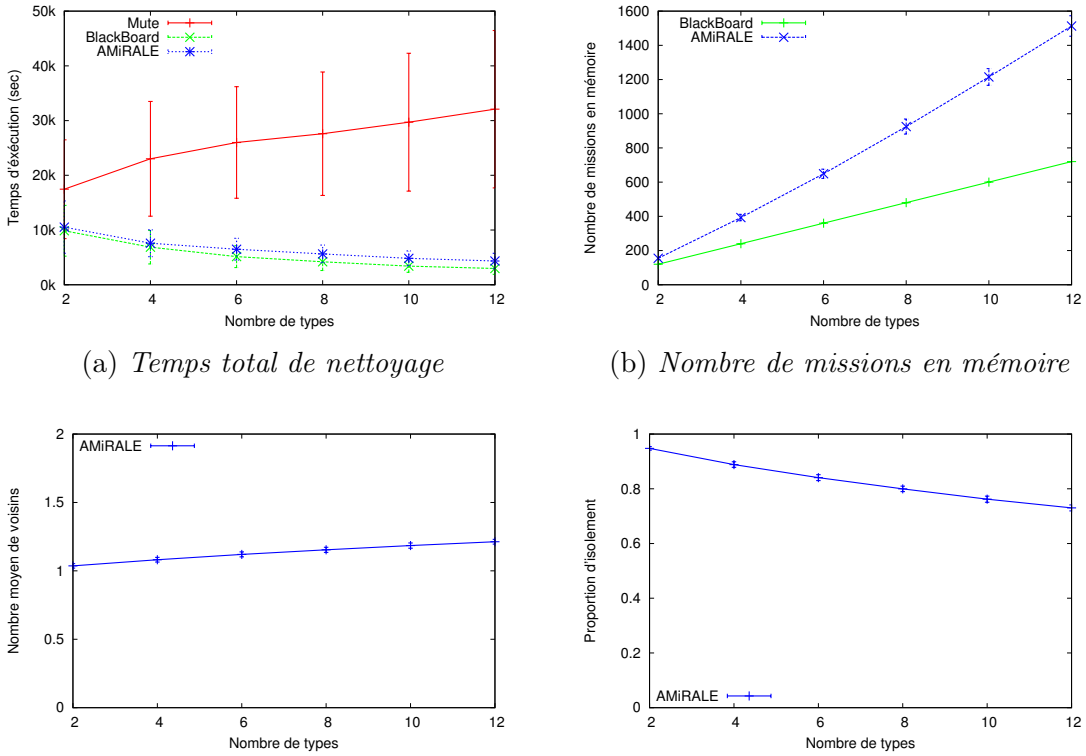
(c) *Nombre de voisins par robot* (d) *Proportion d'isolement par robot*

FIGURE 9.3 – Résultats de simulation pour AMiRALE en temps absolu en fonction du nombre de robots/type.

Nous pouvons constater que l'augmentation du nombre de robots réduit significativement le temps total d'exécution du scénario. Le modèle collectif Mute fournit des résultats médiocres en comparaison de ceux d'AMiRALE et du Blackboard qui sont très similaires. Nous pouvons également remarquer que le bénéfice de l'ajout de nouveaux robots par rapport au temps nécessaire au nettoyage complet du parc est de plus en plus marginal au fur et à mesure que le nombre de robots par type

augmente. La connectivité moyenne est quant à elle meilleure lorsque le nombre de robots est plus important. Ceci permet donc une meilleure diffusion de l'information au sein de la flotte. Enfin, une augmentation du nombre de robots implique une légère augmentation du nombre de missions générées par AMiRALE. Ce phénomène s'explique par le fait que la probabilité pour un robot de détecter un déchet dont il n'a pas connaissance mais qui a déjà été détecté par un autre capteur augmente lorsque le nombre de capteurs potentiels est plus important.

La figure 9.4 utilise comme variable le nombre de types de déchet. Le nombre de déchets par type est fixé à 60 tandis que le nombre de robots par type est fixé à 6. Une augmentation du nombre de types entraîne une augmentation du nombre de déchets ainsi que du nombre de robots, la valeur de ces paramètres étant donnée par type.



(a) *Temps total de nettoyage* (b) *Nombre de missions en mémoire*

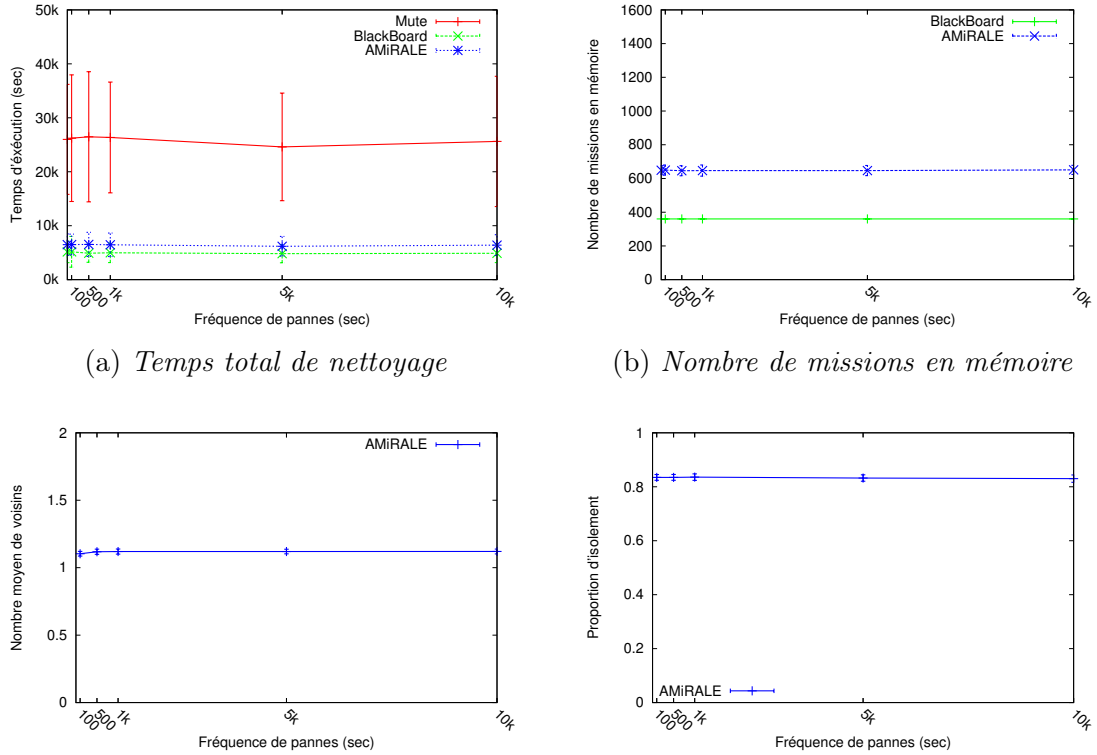
(c) *Nombre de voisins par robot* (d) *Proportion d'isolement par robot*

FIGURE 9.4 – Résultats de simulation pour AMiRALE en temps absolu en fonction du nombre de types de déchet.

Le modèle Mute est de moins en moins performant lorsque le nombre de types augmente. Nous pouvons ainsi affirmer que l'augmentation du nombre de robots qui agissent de manière collective ne permet pas de compenser l'augmentation du nombre de déchets. En revanche, le modèle Blackboard ainsi que notre système AMiRALE fournissent de meilleurs temps de nettoyage lorsque la taille de la flotte augmente même dans le cas où le nombre de déchets à traiter augmente en parallèle. La raison principale est que ces deux méthodes permettent aux robots de connaître l'emplacement de certains déchets compatibles sans avoir besoin de les découvrir par eux-mêmes. L'augmentation du nombre de robots compense donc l'augmentation

du nombre de déchets dans le cas des modèles Blackboard et AMiRALE. Cette affirmation n'est néanmoins pas valable vis à vis du nombre de missions en mémoire. Dans le cas du Blackboard, seul le nombre de déchets a un impact sur le nombre de missions générées. Dans le cas d'AMiRALE, le nombre de robots influe également sur ce paramètre. Par conséquent, une augmentation du nombre de types de déchet influe significativement sur le nombre de missions générées par AMiRALE.

La figure 9.5 s'intéresse à la fréquence de pannes des robots. Le nombre de types ainsi que le nombre de robots par type sont fixés à 6 tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 36 robots et 360 déchets au total). À chaque panne, un robot choisi aléatoirement est supprimé du parc et remplacé par un robot de même type mais dont la mémoire est vide (*i.e.* sans information sur les missions en cours ou terminées).



(a) *Temps total de nettoyage* (b) *Nombre de missions en mémoire*

(c) *Nombre de voisins par robot* (d) *Proportion d'isolement par robot*

FIGURE 9.5 – Résultats de simulation pour AMiRALE en temps absolu en fonction de la fréquence de pannes des robots.

Comme nous pouvions nous y attendre, les effets d'une panne avec le Blackboard sont inexistants dans la mesure où la mémoire des nœuds est partagée. Ainsi, la perte d'un robot n'implique aucune perte d'information. Les effets sur la méthode collective Mute sont légèrement plus marqués. Ces variations ainsi que les valeurs importantes des écart-types sont dues au caractère purement aléatoire des mouvements des robots. Nous pouvons constater que les pannes n'ont qu'un effet négligeable sur les performances de notre système AMiRALE. Ces résultats sont également valables pour des fréquences de pannes élevées (*i.e.* toutes les 100 secondes). Les mécanismes de réplication et de diffusion des missions permettent de limiter la perte d'informa-

tion lors de la suppression d'un robot de la flotte. Nous pouvons ainsi affirmer que notre système AMiRALE est tolérant aux pannes des nœuds. Enfin, nous constatons que la connectivité n'est pas affectée par les pannes des robots.

La figure 9.6 est une fonction de répartition cumulative du temps de ramassage des déchets présents dans le parc. Le nombre de types ainsi que le nombre de robots par type sont fixés à 6 tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 36 robots et 360 déchets au total).

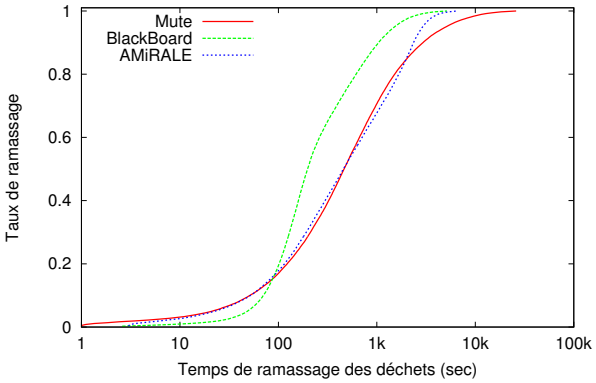


FIGURE 9.6 – Fonction de répartition cumulative de ramassage des déchets pour AMiRALE en temps absolu.

Nous pouvons constater que l'efficacité de notre système AMiRALE est maximal lors du traitement des derniers 20% des déchets.

Conclusion

L'ensemble des résultats présentés dans cette section nous permettent d'émettre plusieurs conclusions sur notre système AMiRALE dans sa version par temps absolu :

- il fournit des résultats proches de ceux du Blackboard en termes de temps de nettoyage ;
- il est compatible avec des flottes fortement déconnectées (*i.e.* rarement connectées et avec peu de voisins) ;
- il est tolérant aux pannes (perte d'un robot et de sa mémoire) même pour des fréquences de pannes élevées ;
- l'augmentation du nombre de robots permet d'augmenter la connectivité de la flotte fournissant ainsi de meilleurs résultats ;
- l'augmentation du nombre de robots entraîne une légère augmentation du nombre de missions générées ;
- le nombre de missions générées augmente linéairement avec le nombre de déchets présents dans le parc ;
- le nombre moyen de missions en mémoire sur l'ensemble des robots est environ deux fois plus élevé que le nombre de déchets présents dans le parc.

Nous pouvons également émettre un résultat théorique sur la consommation mémoire de notre système AMiRALE dans le cadre de ce scénario. Comme nous l'avons expliqué en section 6.2.1, une mission est représentée par un 7-uplet $\{e, k, n, t, s, n', t'\}$ auquel il faut ajouter la donnée de la mission. Le prototype du système AMiRALE

que nous avons développé est programmé en JAVA et utilise les types natifs de ce langage. Le type de déchet e, l'état de la mission s ainsi que les identifiants n et n' sont chacun codés sur 8 bits. Le numéro de séquence k est quant à lui codé sur 32 bits. Les informations temporelles t et t' (*i.e.* dates de création et de dernière modification) sont chacune codées sur 32 bits. Enfin, la donnée de la mission représentant la position géographique du déchet est codée sur 80 bits ; cet encodage permet d'obtenir une précision de l'ordre du mètre en utilisant le système de positionnement GPS [258][259]. Une mission requiert donc 208 bits en mémoire. Nous avons constaté que le nombre de missions générées par AMiRALE est environ égale à deux fois le nombre de déchets présents dans le parc. Par conséquent, pour 720 déchets (nombre maximal de déchets utilisés dans nos expérimentations), le nombre de missions générées serait de l'ordre de 1400, nécessitant ainsi 285 kbit en mémoire. Cet coût est dérisoire en comparaison de la capacité mémoire fournie par la grande majorité des systèmes autonomes actuels.

9.4.2 Impact du filtre f_{blind}

Principe général

Comme nous l'avons expliqué dans la section 6.3.2, les filtres permettent d'adapter certains mécanismes de décisions internes du modèle AMiRALE afin de prendre en compte les spécificités de l'application. Le filtre f_{blind} permet notamment de contrôler la fréquence de diffusion de chaque vue. Dans les simulations précédentes, ce filtre avait pour rôle d'empêcher la diffusion d'une vue dans le cas où la mission relative était terminée depuis plus de 1000 secondes sans contradiction.

Dans cette section, nous allons tester plusieurs configurations du filtre f_{blind} et effectuer les mesures suivantes :

- le temps total nécessaire au nettoyage complet du parc ;
- le nombre moyen de missions en mémoire sur l'ensemble des robots de la flotte en fin de scénario ;
- le nombre moyen de vues par message ;
- le nombre moyen de missions annulées sur la globalité du scénario ;
- les débits moyens en entrée et en sortie pour chaque robot de la flotte.

Les quatre premières mesures seront effectuées par simulation ; les débits seront quant à eux obtenus par émulation à l'aide de NEmu (*c.f.* chapitre 8).

Simulation

Configuration d'AMiRALE

Dans cette section, nous évaluons les performances de notre système AMiRALE avec deux configurations différentes du filtre f_{blind} :

1. empêche la diffusion d'une mission terminée depuis plus de 1000 secondes sans contradiction ;
2. en addition de la configuration 1, empêche la diffusion d'une mission émise et reçue depuis moins de 100 secondes sans contradiction depuis plus de 1000 secondes.

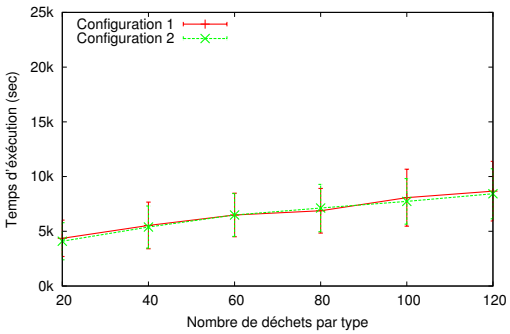
La configuration 2 permet de limiter la bande passante utilisée par chaque robot en diminuant le nombre de vues émises par message. Notre objectif est ici de mesurer l'impact d'une telle configuration sur les performances de notre système AMiRALE.

Paramètres de simulation

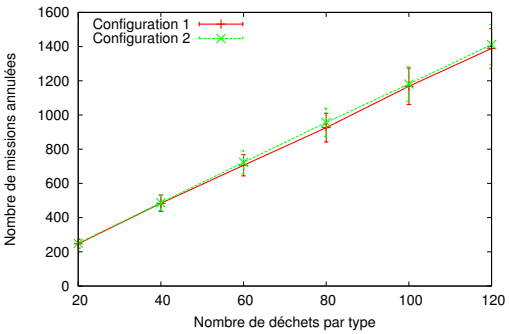
Les paramètres de simulation sont identiques à ceux présentés en section 9.4.1.

Résultats de simulation

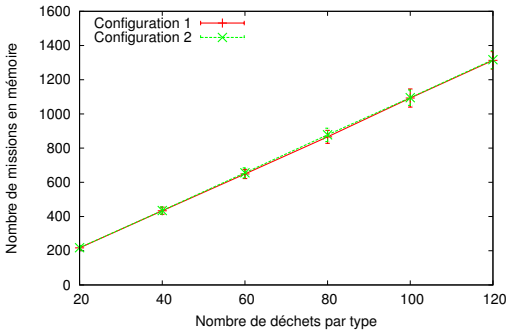
La figure 9.7 présente nos résultats de simulation en fonction du nombre de déchets par type. La figure 9.7a s'intéresse au temps total de nettoyage du parc pour les deux configurations du filtre f_{blind}. La figure 9.7b représente le nombre moyen de missions annulées sur l'ensemble du scénario. Enfin, les figures 9.7c et 9.7d montrent respectivement le nombre moyen de missions en mémoire en fin se scénario ainsi que le nombre moyen de vues par message. Le nombre de types de déchet ainsi que le nombre de robots par type sont fixés à 6 (*i.e.* 36 robots au total).



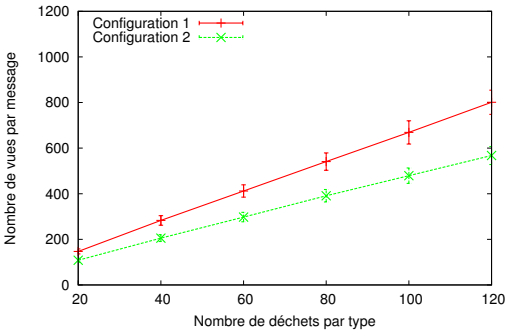
(a) *Temps total de nettoyage*



(b) *Nombre de missions annulées*



(c) *Nombre de missions en mémoire*



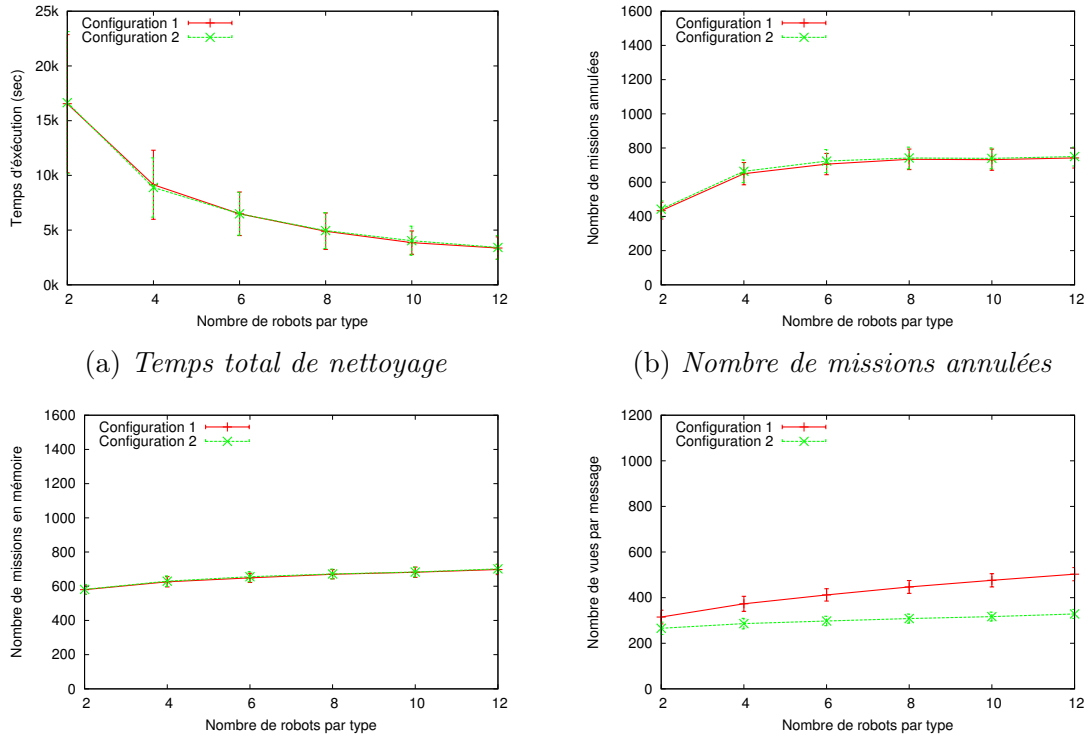
(d) *Nombre de vues par message*

FIGURE 9.7 – Résultats de simulation pour la configuration du filtre f_{blind} en fonction du nombre de déchets/type.

Nous constatons que les deux configurations du filtre f_{blind} fournissent des résultats similaires en termes de rapidité de nettoyage, du nombre de missions en mémoire ainsi que du nombre de missions annulées. Nous soulignons le fait que le nombre de missions annulées est donné pour l'ensemble du scénario alors que le nombre de missions en mémoire est une moyenne calculée sur l'ensemble des robots.

Nous constatons également que la seconde configuration permet de réduire significativement le nombre de vues par message tout en conservant les performances de la première configuration. Enfin, nous remarquons que le nombre de missions annulées augmentent linéairement avec le nombre de déchets présents dans le parc et donc avec le nombre de missions générées. Le nombre de missions annulées sur la totalité du scénario avoisine le nombre moyen de missions en mémoire par robot.

La figure 9.8 s'intéresse aux mêmes indicateurs (*i.e.* temps total de nettoyage, nombre de missions annulées, nombre de missions en mémoire et nombre de vues par message) mais en fonction du nombre de robots par type. Le nombre de types de déchet est fixé à 6, tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 360 déchets au total).



(a) *Temps total de nettoyage* (b) *Nombre de missions annulées*

(c) *Nombre de missions en mémoire* (d) *Nombre de vues par message*

FIGURE 9.8 – Résultats de simulation pour la configuration du filtre f_{blind} en fonction du nombre de robots/type.

Ces résultats confirment deux phénomènes constatés sur la figure 9.7. Premièrement, le nombre de missions annulées sur l'ensemble du scénario est similaire au nombre moyen de missions présentes en mémoire pour chaque robot. Deuxièmement, la seconde configuration du filtre f_{blind} permet de réduire significativement le nombre de vues par message tout en conservant les performances fournies par la première configuration du filtre. En outre, l'augmentation du nombre de robots accroît l'écart du nombre de vues par message entre ces deux configurations. En effet, la seconde configuration du filtre permet de ne pas ré-émettre une vue récemment reçue ou partagée. Comme nous l'avons constaté sur la figure 9.3, une augmentation du nombre de robots implique une augmentation de la connectivité dans la flotte. Ainsi, la probabilité de partager (*i.e.* envoi ou réception) la même version d'une

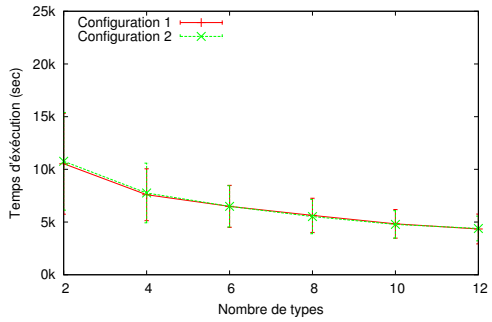
même mission avec le même voisinage augmente. La configuration 2 du filtre f_{blind} permet de prendre en compte ce phénomène et de limiter son impact.

La figure 9.9 utilise comme variable le nombre de types de déchet. Le nombre de déchets par type est fixé à 60 tandis que le nombre de robots par type est fixé à 6. Nous rappelons ici qu'une augmentation du nombre de types entraîne une augmentation du nombre de déchets et du nombre de robots.

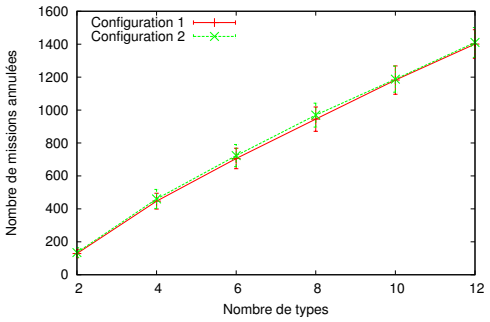
Ici encore, la seconde configuration du filtre f_{blind} permet de diminuer le nombre de vues par message tout en conservant les performances de la première configuration en termes de nombre de missions annulées, de nombre de missions en mémoire et de temps de nettoyage du parc.

La figure 9.10 s'intéresse à la la fréquence de pannes des robots. Le nombre de types ainsi que le nombre de robots par type sont fixés à 6 tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 36 robots et 360 déchets au total).

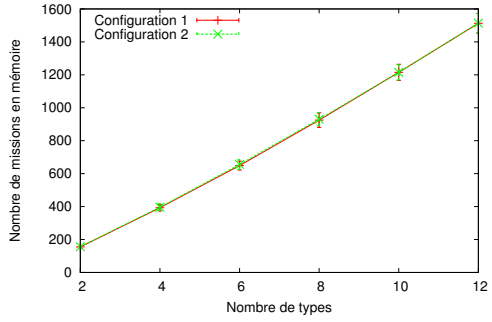
Nous pouvons constater que l'augmentation du taux de pannes des robots entraîne une légère augmentation du nombre moyen de vues par message. Ce phénomène est du à l'insertion systématique d'un nouveau robot dans la flotte à la suite d'une panne rencontré par un autre. Le filtre f_{blind} n'est effectif sur une vue reçue par un robots qu'après les 1000 premières secondes. Les robots diffusent donc de manière soutenue les vues récemment intégrées dans leur mémoire. Un nouveau robot intégrant l'ensemble des vues reçues au début de sa vie (*i.e.* lors de son insertion dans la flotte), il les re-diffuse également lors des 1000 premières secondes après intégration. Une augmentation du taux de pannes impliquant une augmentation du nombre de nouveaux robots, le nombre moyen de vues émises par la flotte augmente.



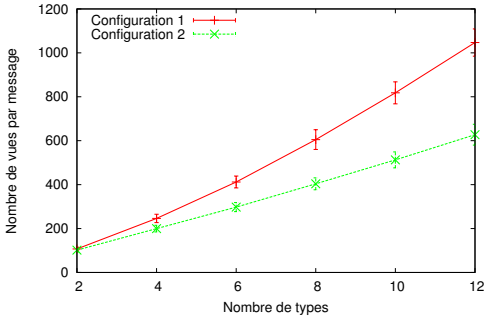
(a) *Temps total de nettoyage*



(b) *Nombre de missions annulées*

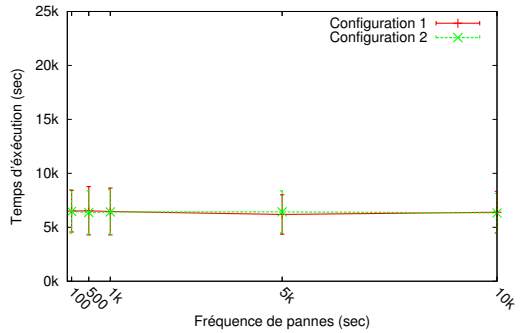


(c) *Nombre de missions en mémoire*

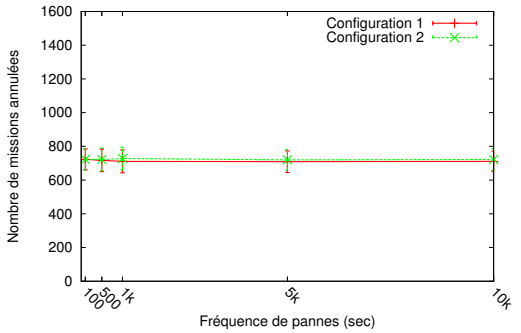


(d) *Nombre de vues par message*

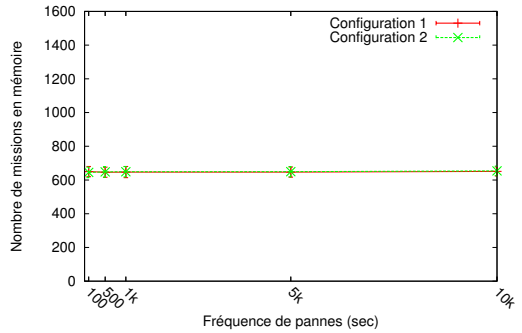
FIGURE 9.9 – Résultats de simulation pour la configuration du filtre f_{blind} en fonction du nombre types de déchet.



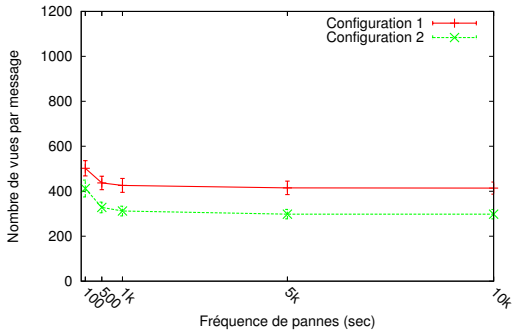
(a) *Temps total de nettoyage*



(b) *Nombre de missions annulées*



(c) *Nombre de missions en mémoire*



(d) *Nombre de vues par message*

FIGURE 9.10 – Résultats de simulation pour la configuration du filtre f_{blind} en fonction de la fréquence de pannes des robots.

Conclusion

L'ensemble des résultats présentés dans cette section nous permet d'émettre de nouvelles conclusions sur notre système AMiRALE :

- une configuration adaptée du filtre f_{blind} permet de réduire le nombre de vues par message, et donc le débit généré, tout en conservant les performances du modèle initial ;
- le nombre de missions annulées sur l'ensemble du scénario de nettoyage augmente linéairement avec le nombre de déchets présents dans le parc ;
- le nombre de missions annulées sur l'ensemble du scénario est similaire au nombre moyen de missions présentes en mémoire par robot en fin de scénario ;
- une fréquence importante de remplacement des robots implique une augmentation du nombre de vues par message.

Nous pouvons également émettre un résultat théorique sur la bande passante requise par notre système AMiRALE dans le cadre de ce scénario de nettoyage de parc. Nous avons montré à la section 9.4.1 qu'une mission requiert 208 bits en mémoire. Une vue, dans la version par temps absolu du modèle, étant équivalente à une mission, celle-ci est également codée sur 208 bits. Comme nous l'avons évoqué dans le section 6.2.2, les vues sont agrégées en un message unique contenant l'identifiant de l'émetteur du message ainsi que des données supplémentaires optionnelles. Dans le cadre de notre scénario de nettoyage de parc, cette donnée optionnelle contient la position géographique du nœud émetteur du message. Par conséquent, la taille d'un

message contenant n vues est égale à $208n + 88$ bits. Le nombre de vues par message atteint lors de nos simulations est de 1050 pour la première configuration et de 630 pour la seconde (720 déchets et 36 robots au total). Par conséquent, la taille moyenne des messages est respectivement de 214 kbit et de 128 kbit. Un message étant émis toute les 5 secondes, la première configuration produit un débit sortant de 43 kbit/s tandis que la seconde ne nécessite que 26 kbit/s. Ces débits sont compatibles avec les technologies réseaux utilisées dans le domaine des systèmes autonomes présentées dans la section 2.1.

Émulation

Configuration d'AMiRALE

En complément des résultats de simulation présentés dans cette section, nous fournissons des résultats d'expérimentation en environnement virtuel portant sur la consommation réseau de notre modèle AMiRALE. Ces expérimentations ont été réalisées avec notre émulateur NEmu présenté dans le chapitre 8. Le prototype de notre système AMiRALE utilisé dans cette expérimentation est le même que celui utilisé pour l'ensemble des simulations effectuées dans ce chapitre. L'application JAVA utilisée dans cette expérimentation diffère sur deux points de celle utilisée dans les simulations :

— La gestion du temps est continue ; JbotSim fonctionne quant à lui en temps discret.

— Les communications réseaux sont faites au travers de *broadcast* UDP. JbotSim ne repose pas sur une pile réseau réelle. En effet, les communications effectuées entre les nœuds reposent sur le mécanisme de passage d'objets par référence fourni par le langage JAVA.

L'objectif ce cette expérimentation est de mesurer les débits moyens réels en entrée et en sortie des nœuds. Pour cela, nous avons appliqué trois configurations différentes au filtre f_{blind} :

1. empêche la diffusion des vues relatives aux missions terminées depuis plus de 10 secondes ;
2. empêche la diffusion des vues relatives aux missions terminées depuis plus de 60 secondes ;
3. n'applique aucune restriction.

Paramètres d'émulation

Dans cette expérimentation, chaque machine virtuelle se déplace aléatoirement dans un espace rectangulaire à deux dimensions. Chaque machine virtuelle est à la fois capteur et solveur pour tous les types de mission. Les missions sont créées, verrouillées et résolues à intervalles de temps pseudo-aléatoires. Le scénario prend fin lorsque la mémoire de chaque nœud contient un nombre seuil de missions terminées. Les paramètres de notre expérimentation sont décrits dans le tableau 9.2.

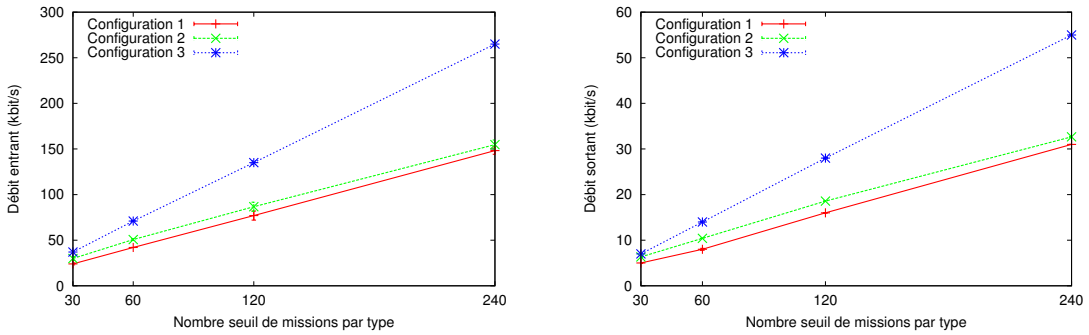
TABLE 9.2 – Paramètres d'émulation.

Paramètre	**Valeur**
Émulateur	NEmu
Taille du terrain	1000 m x 1000 m
Taille des nœuds	1 m x 1 m
Vitesse des nœuds	5 m / s
Modèle de mobilité	*Random Waypoint*
Nombre de nœuds	6
Nombre de types	6
Fréquence des communications	5 s
Modèle de propagation	*Two-ray ground reflection*
Mémoire vive des nœuds	512 Mo
Carte réseau des nœuds	Intel e1000

Résultats d'émulation

La figure 9.11a présente la moyenne du débit sortant des nœuds (*i.e.* débit des messages sortants par nœud) en fonction du nombre seuil de missions par type. Nous pouvons constater que l'évolution du débit sortant par nœud augmente linéairement avec le nombre de missions. Nous pouvons également remarquer qu'une configuration plus fine du filtre f_{blind} permet de réduire significativement le débit sortant par nœud. Ce nouveau résultat est donc une confirmation des conclusions émises dans la partie simulation de cette section.

La figure 9.11b présente la moyenne du débit entrant des nœuds (*i.e.* débit des messages entrants par nœud) en fonction du nombre seuil de missions par type. Les résultats sont similaires à ceux présentés en figure 9.11a mais avec des débits plus élevés. Pour un nombre important de missions (*i.e.* 240 par type, soit 1440 missions au total) le débit entrant avoisine les 270 kbit/s pour la troisième configuration (*i.e.* sans utilisation du filtre f_{blind}). Une configuration plus fine de ce filtre permet d'obtenir un débit entrant avoisinant les 160 kbit/s. Ce débit est compatible avec les principales technologies de communication utilisées pour les flottes autonomes (*i.e.* Bluetooth, ZigBee et Wi-Fi ; *c.f.* section 2.1). Il est également important de noter que l'implémentation d'AMiRALE utilisée contient des champs d'au moins 1 octet. Une implémentation plus fine, au niveau bit, permettrait de diminuer la taille de certains champs (*e.g.* état de la mission, identifiant des nœuds) et donc de diminuer la bande passante nécessaire.



(a) *Débit sortant* (b) *Débit entrant*

FIGURE 9.11 – Résultats d'émulation pour la configuration du filtre f_{blind} en fonction du nombre seuils de missions par type.

Conclusion

Les résultats d'émulation confirment l'impact du filtre f_{blind} sur les débits entrants et sortants par nœud. Ces expérimentations permettent de justifier l'existence des filtres du modèle AMiRALE et soulignent leur importance. Grâce au filtre f_{blind}, AMiRALE est compatible avec les principales technologies de communication utilisées pour les systèmes autonomes.

9.4.3 Comparaison des approches temporelles

Principe général

Dans cette section, nous souhaitons comparer les performances des différentes approches temporelles de notre modèle AMiRALE présentées dans la partie II de ce document. Comme nous l'avons expliqué dans la section section 6.4, la différence de configuration initiale ainsi que la dérive des horloges doivent être prises en compte dans les communications entre les nœuds de la flotte. C'est la raison pour laquelle nous avons développé trois versions distinctes de notre modèle collaboratif AMiRALE répondant chacune à une stratégie de synchronisation temporelle particulière :

— l'approche par temps absolu où le contenu des missions et des vues sont identiques, les horloges étant synchronisées par un système global ;
— l'approche par temps relatif où les dates t et t' présentes dans les vues sont traduites en intervalles de temps relatifs à l'horloge de l'émetteur ;
— l'approche par ordre relatif où aucune information temporelle n'est partagée dans les vues ; seule l'identifiant des missions ainsi que leur date de réception permettent de les ordonner.

Dans les expérimentations précédentes, nous nous sommes exclusivement concentrés sur l'approche par temps absolu. Cette section a pour objectif de la comparer aux deux autres versions.

Configuration d'AMiRALE

La configuration d'AMiRALE est identique à celle utilisée lors des simulations précédentes. Le filtre f_{blind} empêche la diffusion d'une mission terminée depuis plus de 1000 secondes sans contradiction. Il empêche également la diffusion d'une mission émise et reçue depuis moins de 100 secondes sans contradiction depuis plus de 1000 secondes.

Paramètres de simulation

Les paramètres de simulation sont identiques à ceux utilisés dans les simulations précédentes. Pour les approches par temps relatif et ordre relatif de notre modèle, nous avons désynchronisé les horloges de chaque robot d'au minimum une seconde. En d'autres termes, pour tous robots x et y de la flotte et à tout moment, la différence de valeur de leur horloge respective est au minium de 1 seconde. Ces approches

temporelles étant destinées à une flotte d'entités dont les horloges ne sont pas synchronisées, cette configuration nous permet de simuler un environnement réaliste.

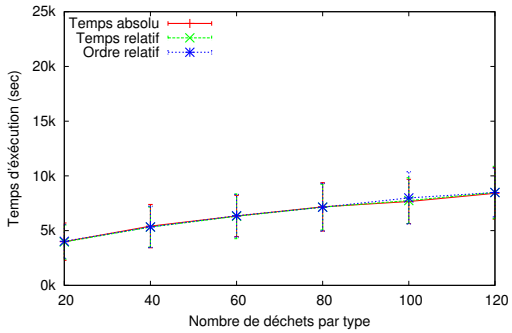
Pour chacune des approches temporelles de notre système AMiRALE, nous effectuons les mesures suivantes :

— le temps total nécessaire au nettoyage complet du parc ;
— le nombre moyen de missions en mémoire sur l'ensemble des robots de la flotte en fin de scénario ;
— le nombre moyen de vues par message ;
— le nombre moyen de missions annulées sur la globalité du scénario.

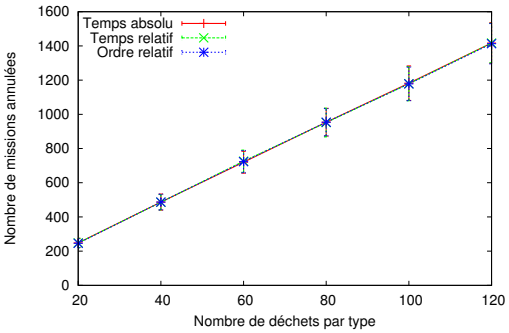
Les paramètres sont les mêmes que ceux utilisés dans les sections 9.4.1 et 9.4.2 (*i.e.* nombre de types de déchet, nombre de déchets par type, nombre de robots par type et fréquence de pannes des robots). Enfin, nous comparons les performances des trois versions de notre modèle en faisant varier la durée de validité des verrous (*i.e.* Ψ_{will} et Ψ_{do}).

Résultats de simulation

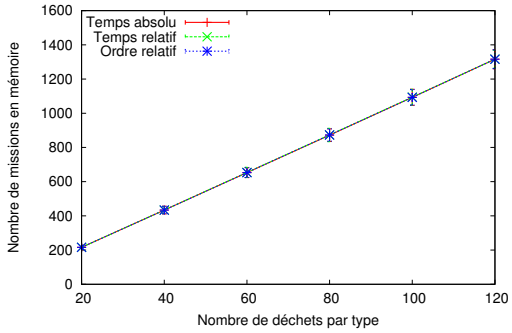
La figure 9.12 présente nos résultats de simulation en fonction du nombre de déchets par type. La figure 9.12a s'intéresse au temps total de nettoyage du parc pour les trois approches temporelles de notre système AMiRALE. La figure 9.12b illustre le nombre moyen de missions annulées sur l'ensemble du scénario. Enfin, les figures 9.12c et 9.12d montrent respectivement le nombre moyen de missions en mémoire par robot en fin de scénario ainsi que le nombre moyen de vues par message. Le nombre de types de déchet ainsi que le nombre de robots par type sont fixés à 6 (*i.e.* 36 robots au total).



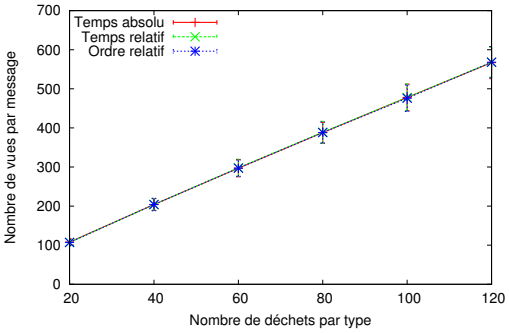
(a) *Temps total de nettoyage*



(b) *Nombre de missions annulées*



(c) *Nombre de missions en mémoire*

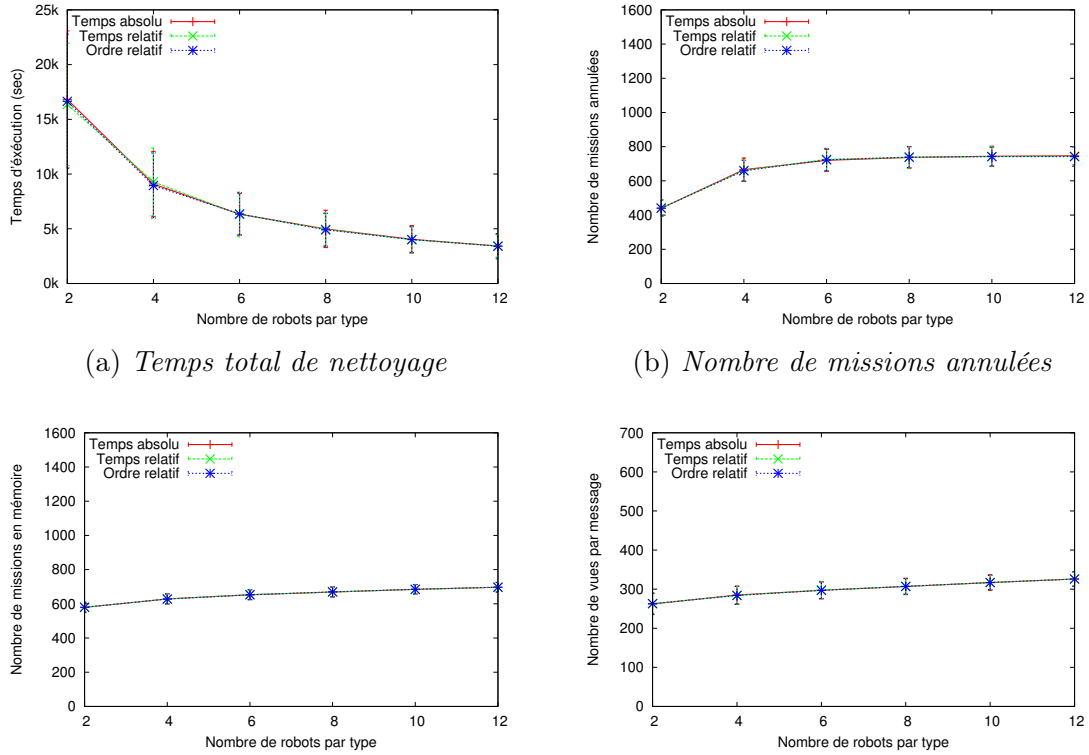


(d) *Nombre de vues par message*

FIGURE 9.12 – Résultats de simulation pour chacune des approches temporelles en fonction du nombre de déchets/type.

Nous pouvons constater que les trois versions de notre modèle fournissent des résultats similaires lorsque le nombre de déchets augmente. Le nombre de missions n'est donc pas un critère décisif dans le choix d'une des stratégies de synchronisation de notre modèle.

La figure 9.13 s'intéresse aux mêmes indicateurs (*i.e.* temps total de nettoyage, nombre de missions annulées, nombre de missions en mémoire et nombre de vues par message) mais en fonction du nombre de robots par type. Le nombre de types de déchet est fixé à 6, tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 360 déchets au total).



(a) *Temps total de nettoyage* (b) *Nombre de missions annulées*

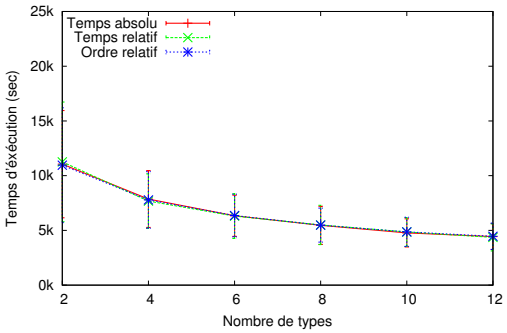
(c) *Nombre de missions en mémoire* (d) *Nombre de vues par message*

FIGURE 9.13 – Résultats de simulation pour chacune des approches temporelles en fonction du nombre de robots/type.

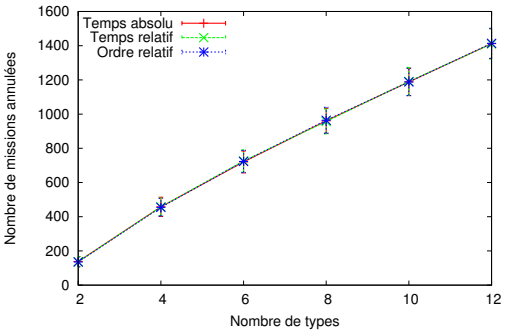
Nous pouvons également constater que la variation du nombre de robots dans la flotte ne permet pas de déterminer la meilleure approche temporelle pour notre modèle collaboratif dans le cadre de ce scénario.

La figure 9.14 utilise comme variable le nombre de types de déchet. Le nombre de déchets par type est fixé à 60 tandis que le nombre de robots par type est fixé à 6. Nous rappelons qu'une augmentation du nombre de types entraîne une augmentation du nombre de déchets ainsi que du nombre de robots.

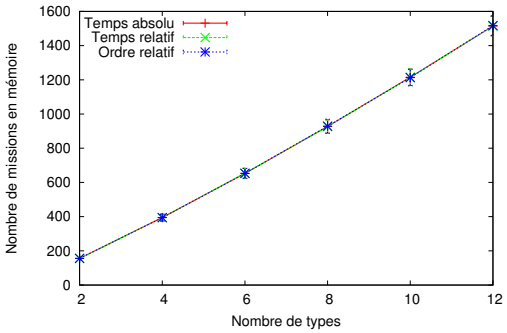
Comme nous pouvions le prévoir à la vue des résultats des deux précédentes simulations, le nombre de types de missions n'est pas un élément déterminant dans le choix d'une approche temporelle dans le cadre de ce scénario.



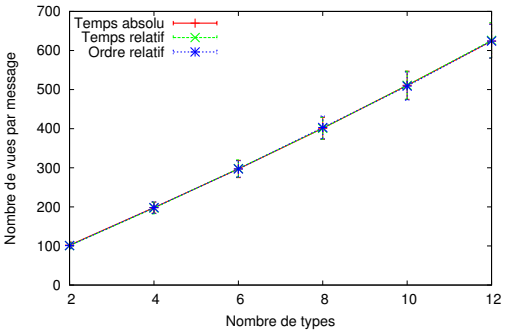
(a) *Temps total de nettoyage*



(b) *Nombre de missions annulées*

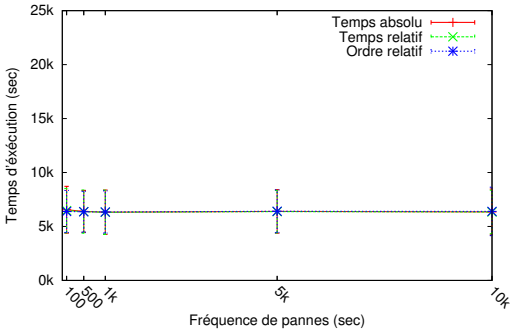


(c) *Nombre de missions en mémoire*

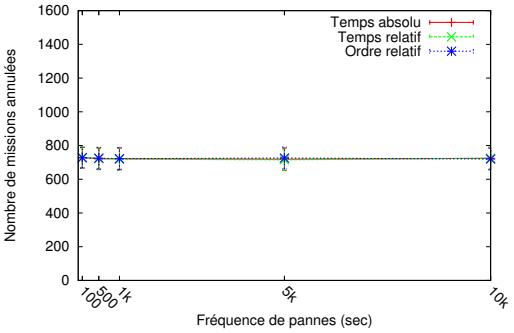


(d) *Nombre de vues par message*

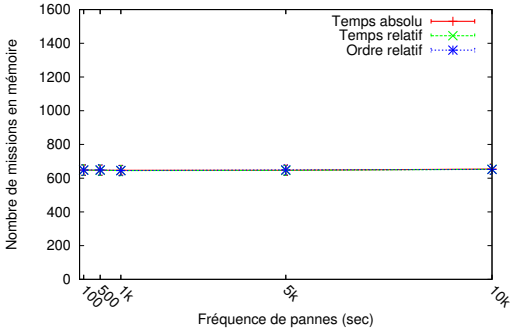
FIGURE 9.14 – Résultats de simulation pour chacune des approches temporelles en fonction du nombre de types de déchet.



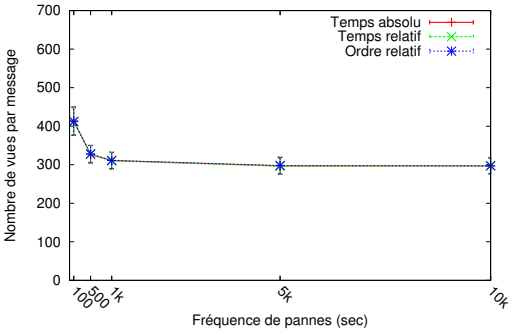
(a) *Temps total de nettoyage*



(b) *Nombre de missions annulées*



(c) *Nombre de missions en mémoire*



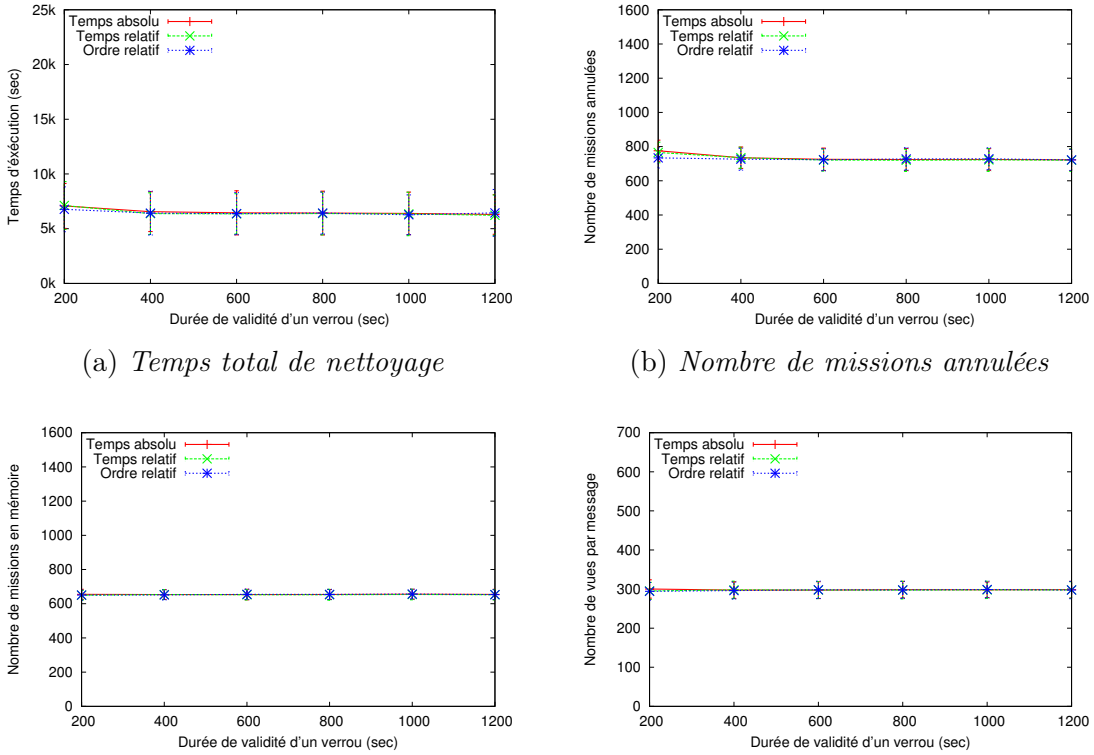
(d) *Nombre de vues par message*

FIGURE 9.15 – Résultats de simulation pour chacune des approches temporelles en fonction de la fréquence de pannes des robots.

La figure 9.15 s'intéresse à la fréquence de pannes des robots. Le nombre de types ainsi que le nombre de robots par type sont fixés à 6 tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 36 robots et 360 déchets au total).

Dans cette simulation, les trois versions de notre modèle fournissent également des performances équivalentes.

Dans la figure 9.16, nous utilisons la durée de validité des verrous comme paramètre de simulation. Les trois versions de notre modèle reposant sur des approches différentes dans la gestion des informations temporelles, nous souhaitons évaluer l'impact de la durée des verrous sur ces différentes versions d'AMiRALE.



(a) *Temps total de nettoyage* (b) *Nombre de missions annulées*

(c) *Nombre de missions en mémoire* (d) *Nombre de vues par message*

FIGURE 9.16 – Résultats de simulation pour chacune des approches temporelles en fonction de la durée de validité des verrous.

Nous constatons que la durée de validité des verrous n'a qu'un impact négligeable sur les performances des trois versions de notre système. Une légère différence de performance entre les trois approches apparaît pour une durée faible de validité des verrous (*i.e.* 200 secondes). En effet, l'approche par ordre relatif est légèrement plus rapide que les deux autres approches (*i.e.* temps relatif et temps absolu). Ce résultat est appuyé par un nombre de missions annulées légèrement moins important que ceux produits par les deux autres approches. Ce phénomène s'explique par le fait que le modèle par ordre relatif évalue la durée des verrous par rapport à la date de réception de la vue associée. Prenons par exemple le cas d'une mission $m_e^{n:k}$ verrouillée en *will* à la date t par le solveur x. Si le solveur y, n'ayant pas connaissance de l'existence de la mission $m_e^{n:k}$, reçoit une vue relative à celle-ci à la date $t' = t + 30$:

— Dans le cas de l'approche par ordre relatif : le solveur y ne pourra re-verrouiller la mission $m_e^{n:k}$ qu'après $t + \Psi_{will} + 30$ secondes.
— Dans le cas des deux autres approches : le solveur y ne devra attendre que $t + \Psi_{will}$ secondes pour pouvoir re-verrouiller la mission $m_e^{n:k}$.

Le temps d'attente pour un solveur avant de pouvoir re-verrouiller une mission est donc en moyenne plus élevée dans le cadre de l'approche par ordre relatif. Plus la durée de validité d'un verrou est faible, plus la probabilité qu'un solveur n'ai pas assez de temps pour résoudre une mission dans le temps imparti augmente. Par conséquent, la probabilité de re-verrouillage et donc de conflit sur une même mission augmente. L'évaluation de la durée des verrous étant plus généreuse dans le modèle par ordre relatif, le nombre de re-verrouillages et donc de missions annulées est légèrement inférieur à ceux des deux autres approches.

Conclusion

Dans cette section, nous avons pu montrer que les trois approches temporelles de notre système AMiRALE fournissent des performances similaires en termes de temps de nettoyage, de nombre moyen de missions en mémoire, de nombre moyen de vues par message ainsi que de nombre de missions annulées sur l'ensemble du scénario. Nous avons également remarqué que l'approche par ordre relatif est légèrement plus performante que les deux autres en termes de temps de nettoyage ainsi que du nombre de missions annulées lorsque la durée de validité des verrous est faible.

9.4.4 Impact de la préemption asynchrone

Principe général

Dans cette section, nous souhaitons évaluer l'impact de l'utilisation de la règle de préemption asynchrone présentée en section 7.3.5. Celle-ci permet à un robot de traiter un déchet à proximité et de modifier le statut de la mission qui lui est associée même si celle-ci est verrouillée par un autre solveur. Sans préemption, un robot ne peut en aucun cas traiter un déchet dont la mission associée n'est pas verrouillée par lui même. Notre objectif est d'évaluer l'impact de cette préemption sur notre scénario de nettoyage de parc.

Configuration d'AMiRALE

La configuration d'AMiRALE est identique à celle utilisée dans la section précédente.

Paramètres de simulation

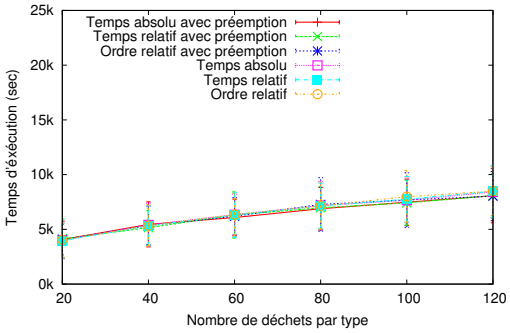
Les paramètres de simulation sont identiques à ceux utilisés dans la section précédente. Pour chacune des approches temporelles de notre système AMiRALE, nous effectuons les mesures suivantes :

— le temps total nécessaire au nettoyage complet du parc ;

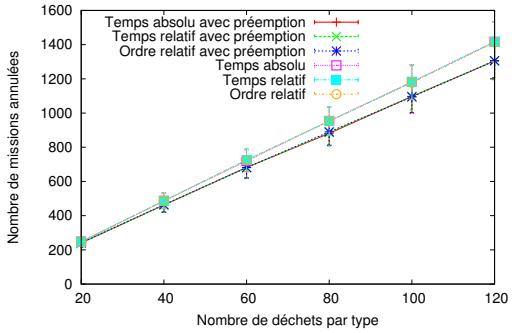
— le nombre moyen de missions annulées sur la globalité du scénario ;
— le nombre moyen de missions préemptées sur la globalité du scénario.

Résultats de simulation

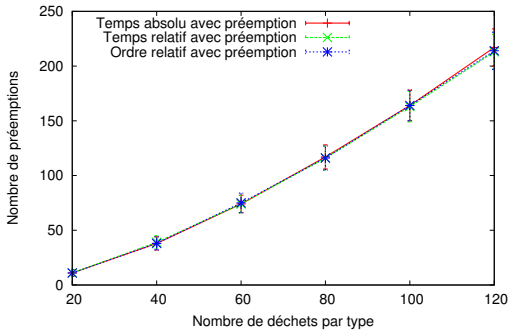
La figure 9.17 présente nos résultats de simulation en fonction du nombre de déchets par type. La figure 9.17a s'intéresse au temps total de nettoyage du parc pour les trois approches temporelles de notre modèle AMiRALE avec et sans préemption. La figure 9.17b illustre le nombre moyen de missions annulées sur l'ensemble du scénario. Enfin, la figures 9.17c montre le nombre de missions préemptées en fin scénario. Le nombre de types de déchet ainsi que le nombre de robots par type sont fixés à 6 (*i.e.* 36 robots au total).



(a) *Temps total de nettoyage*



(b) *Nombre de missions annulées*



(c) *Nombre de missions préemptées*

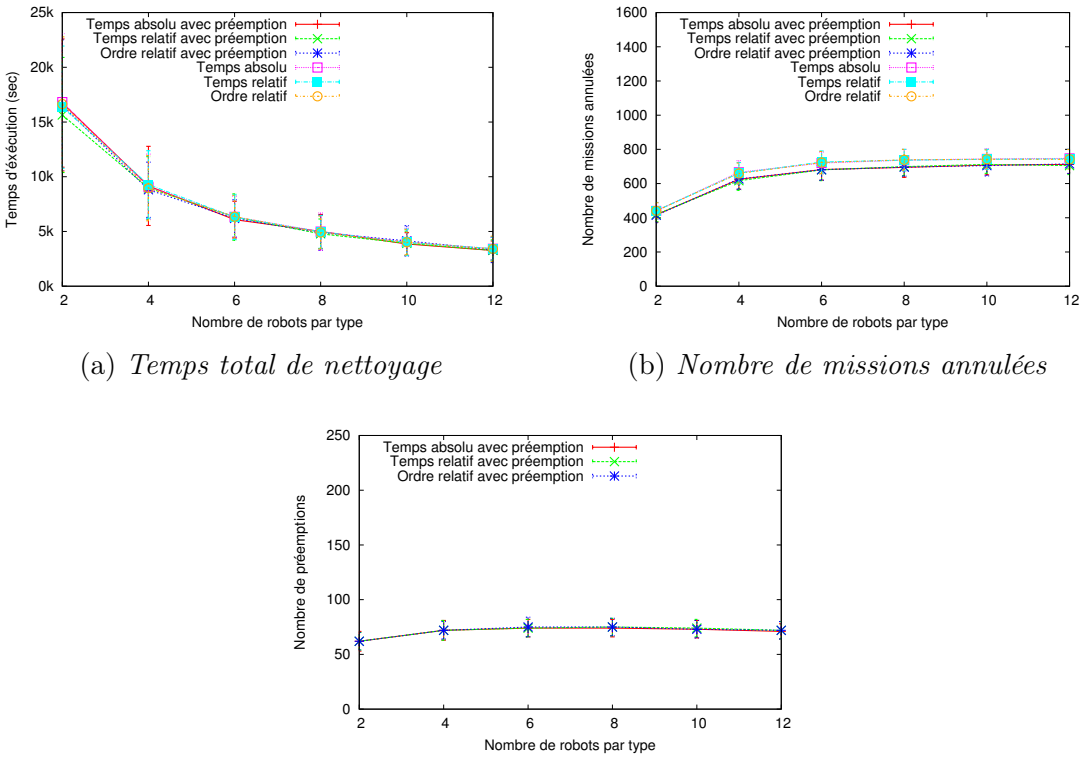
FIGURE 9.17 – Résultats de simulation pour chacune des approches temporelles avec ou sans préemption en fonction du nombre de déchets/type.

Comme nous pouvions le prévoir, le nombre de préemptions augmentent lorsque le nombre de déchets présents dans le parc est plus important. La préemption permet de diminuer légèrement le nombre de missions annulées sur l'ensemble du scénario. L'écart du nombre de missions annulées pour l'ensemble des approches avec et sans préemption est proportionnelle au nombre de missions préemptées. En revanche, la préemption n'a qu'un effet négligeable sur le temps total de nettoyage du parc.

La figure 9.18 s'intéresse aux mêmes indicateurs (*i.e.* temps total de nettoyage, nombre de missions annulées et nombre de missions préemptées) en fonction du

nombre de robots par type. Le nombre de types de déchet est fixé à 6, tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 360 déchets au total).

Nous constatons que le nombre de robots a un impact négligeable sur le nombre de préemption. Tout comme pour la simulation précédente, la préemption ne permet pas de diminuer le temps total de nettoyage du parc. En revanche, elle permet de diminuer légèrement le nombre de missions annulées. Enfin, l'écart du nombre de missions annulées pour l'ensemble des approches avec et sans préemption est également proportionnelle au nombre de missions préemptées.



(a) *Temps total de nettoyage*

(b) *Nombre de missions annulées*

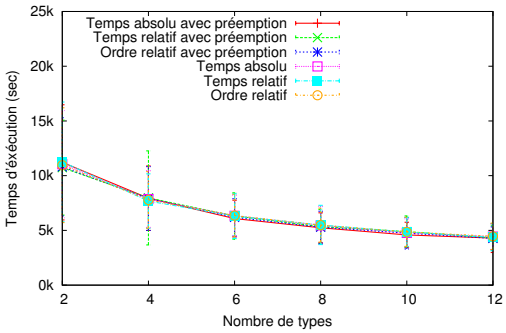
(c) *Nombre de missions préemptées*

FIGURE 9.18 – Résultats de simulation pour chacune des approches temporelles avec ou sans préemption en fonction du nombre de robots/type.

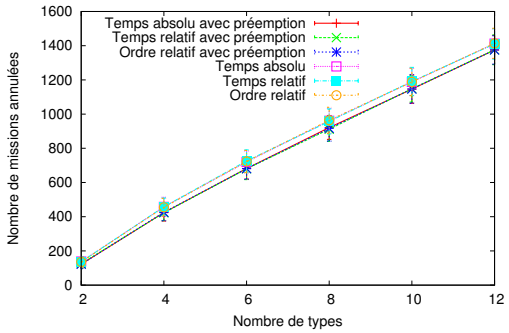
La figure 9.19 utilise comme variable le nombre de types de déchet. Le nombre de déchets par type est fixé à 60 tandis que le nombre de robots par type est fixé à 6. Nous rappelons qu'une augmentation du nombre de types entraîne une augmentation du nombre de déchets ainsi que du nombre de robots.

Le nombre de types augmentant à la fois le nombre de déchets présents dans le parc ainsi que le nombre de robots, les mêmes phénomènes remarqués sur les deux figures précédentes sont également visibles sur celle-ci. L'apport de la préemption sur le nombre de missions annulées semble néanmoins plus faible.

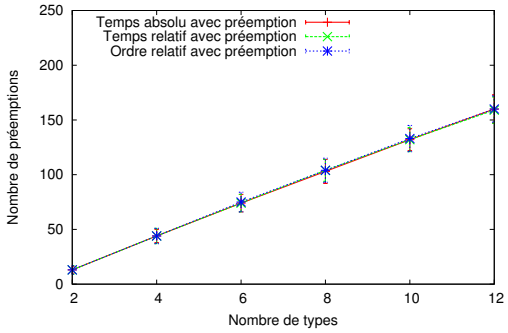
La figure 9.20 s'intéresse à la fréquence de pannes des robots. Le nombre de types ainsi que le nombre de robots par type sont fixés à 6 tandis que le nombre de déchets par type est fixé à 60 (*i.e.* 36 robots et 360 déchets au total).



(a) *Temps total de nettoyage*

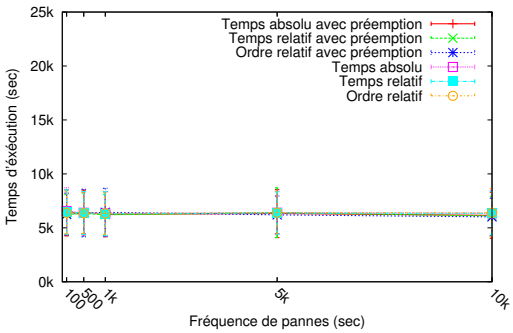


(b) *Nombre de missions annulées*

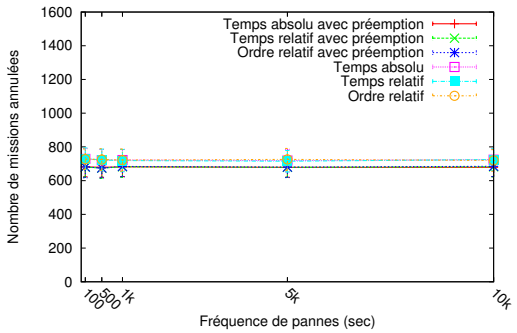


(c) *Nombre de missions préemptées*

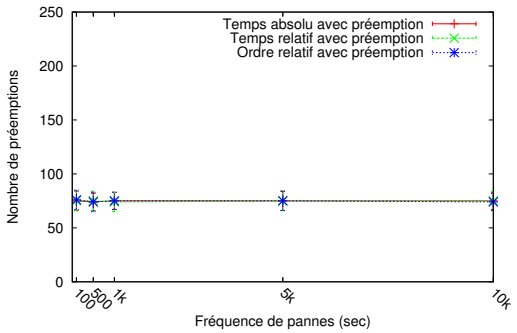
FIGURE 9.19 – Résultats de simulation pour chacune des approches temporelles avec ou sans préemption en fonction du nombre de types de déchet.



(a) *Temps total de nettoyage*



(b) *Nombre de missions annulées*



(c) *Nombre de missions préemptées*

FIGURE 9.20 – Résultats de simulation pour chacune des approches temporelles avec ou sans préemption en fonction de la fréquence de pannes des robots.

Nous pouvons constater que la fréquence de pannes des robots n'a aucune implication sur les effets de la préemption. En effet, la préemption permet de diminuer légèrement le nombre de missions annulées mais n'a qu'un impact négligeable sur le temps total d'exécution du scénario. La fréquence de pannes n'a pas d'influence sur le nombre de préemptions.

Conclusion

Dans cette section, nous avons pu constater que la préemption permet de réduire légèrement le nombre de missions annulées sur la totalité de notre scénario de nettoyage de parc. Nous avons également remarqué que le nombre de missions préemptées évoluent proportionnellement avec le nombre de mission annulées et donc de déchets présents dans le parc. De plus, l'écart du nombre de missions annulées pour l'ensemble des approches temporelles avec et sans préemption semble être proportionnelle au nombre de missions préemptées. Enfin, nous avons montré que la préemption ne permet pas de réduire le temps nécessaire au nettoyage du parc.

9.4.5 Influence du modèle de mobilité

Principe général

Comme nous l'avons expliqué dans le chapitre 3, les modèles de mobilité ont une importance significative sur les performances des protocoles réseaux et des algorithmes distribués. Dans cette section, nous nous intéressons donc à l'impact du modèle de mobilité utilisé par la flotte exploitant notre système collaboratif AMiRALE. Pour ce faire, nous utilisons la version aérienne de notre scénario. Dans celle-ci, des drones aériens sont utilisés pour repérer les déchets au sol depuis les airs et transmettre leur position aux robots terrestres chargés de les collecter. Les drones sont donc des $Sens_i$ pour tout type i de détritus. Chaque robot est $Solv_i$ pour i un type particulier de déchet. Dans cette version du scénario, un robot ne peut détecter que les déchets de son propre type, *i.e.* chaque robot est à la fois $Sens_i$ et $Solv_i$ pour un seul types de déchet i.

Notre objectif est ici de comparer les performances d'AMiRALE en termes de temps de nettoyage et de dépense énergétique (*i.e.* distance moyenne parcourue par les robots, l'utilisation des moteurs étant une des principales sources de consommation [257]), de quantité mémoire utilisée, du nombres de vues par messages ainsi que du nombre de missions annulées. Ces résultats sont obtenus en fonction du nombre de déchets par type présents dans le parc. Un robot ne se déplace que s'il est actif pour une mission donnée (*i.e.* il possède un verrou sur une mission en *will* ou en *do*). Les drones se déplacent en permanence selon trois modèles de mobilité différents que nous avons présentés dans le chapitre 3 :

— *Random Waypoint* (*c.f.* section 3.2.1) où les drones se déplacent de manière aléatoire en sélectionnant des points de destination arbitraires dans le plan ;
— *Smooth Turn* (*c.f.* section 3.3.2) où les nœuds se déplacent de manière circulaire autour d'un centre virtuel pris pseudo-aléatoirement pendant une durée aléatoire bornée ;

— *Semi-Random Circular Movement* (*c.f.* section 3.3.2) où les drones se déplacent de manière circulaire autour du centre du plan en changeant aléatoirement la taille du rayon à chaque tour.

Le *Smooth Turn* et le *Semi-Random Circular Movement* sont deux modèles de mobilité qui prennent en compte les particularités aérodynamiques des engins volants. Nous mesurons tout d'abord le nombre moyen de voisins par nœuds sur l'ensemble de la simulation ainsi que la proportion moyenne de temps pour laquelle les nœuds sont complètement isolés. Ces résultats sont fournis pour chacun des modèles de mobilité présentés ci-dessus.

Paramètres de simulation

Nos paramètres de simulation sont décrits dans le tableau 9.3.

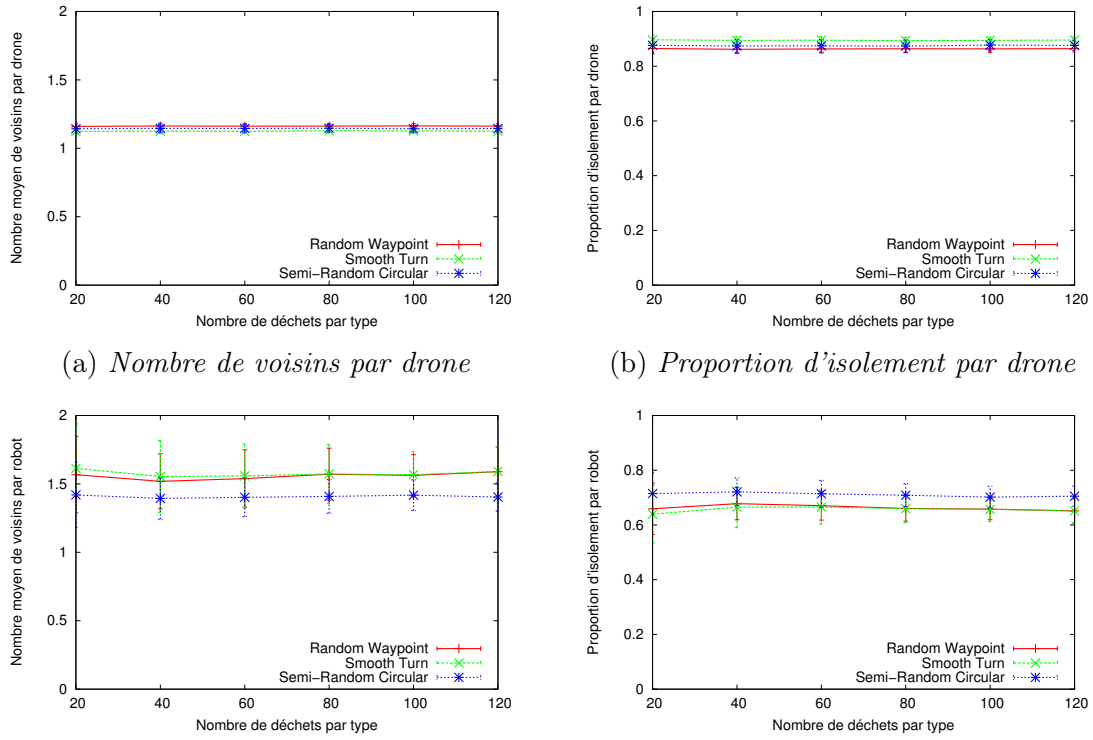
TABLE 9.3 – Paramètres de simulation de la déclinaison aérienne du scénario ParCS.

Paramètre	**Valeur**
Simulateur	JBotSim
Taille du parc	1000 m x 1000 m
Taille des robots / drones	1 m x 1 m
Vitesse des robots / drones	5 m / s
Fréquence des communications	5 s
Portée de communication	30 m
Portée de détection des robots	30 m
Portée de détection des drones	90 m
Nombre de types de déchet	6
Nombre de robots par type	6
Nombre total de robots	36
Nombre total de drones	6
Nombre d'exécutions par point	500

Résultats de simulation

La figure 9.21 présente nos résultats de connectivité (*i.e.* nombre moyen de voisins et taux moyen d'isolement) des nœuds en fonction du nombre de déchets par type.

Nous pouvons constater que la connectivité moyenne par drone offerte par chacun des modèles de mobilité est similaire. En moyenne, un drone est isolé des autres nœuds (*i.e.* drones ou robots) de la flotte entre 80 et 90% du temps. Lors des rares communications qu'il peut effectuer, son voisinage se limite en moyenne à un seul nœud. La connectivité des robots est quant à elle plus élevée avec en moyenne entre un et deux voisins lorsque les robots ne sont pas isolés. Cette connectivité est plus faible pour le modèle *Semi-Random Circular*. Comme nous l'avons évoqué dans les chapitres 2 et 3, la connectivité a un impact significatif sur les performances des algorithmes distribués dans les réseaux mobiles. Par conséquent, nos résultats de performance devraient être en retrait dans le cas du modèle de mobilité *Semi-Random Circular*.



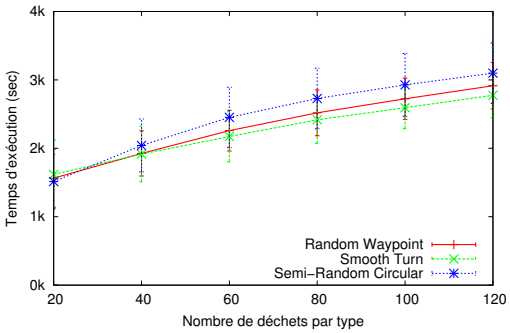
(a) *Nombre de voisins par drone* (b) *Proportion d'isolement par drone*

(c) *Nombre de voisins par robot* (d) *Proportion d'isolement par robot*

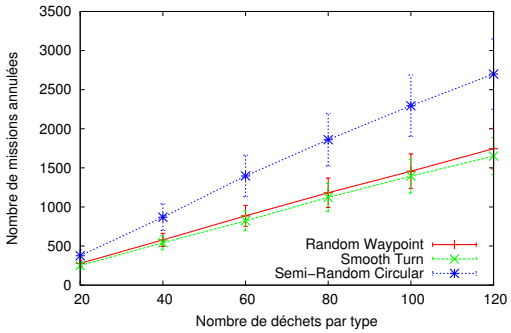
FIGURE 9.21 – Résultats de simulation de connectivité pour AMiRALE en temps absolu en fonction du modèle de mobilité des drones.

La figure 9.22a montre le temps total de nettoyage du parc pour les différents modèles de mobilité des drones. La figure 9.22b illustre le nombre moyen de missions annulées sur l'ensemble du scénario. Les figures 9.22c et 9.22d montrent respectivement le nombre moyen de missions en mémoire pour chaque robot en fin de scénario ainsi que le nombre moyen de vues par message. Enfin, la figure 9.22e s'intéresse à la distance moyenne parcourue par robot.

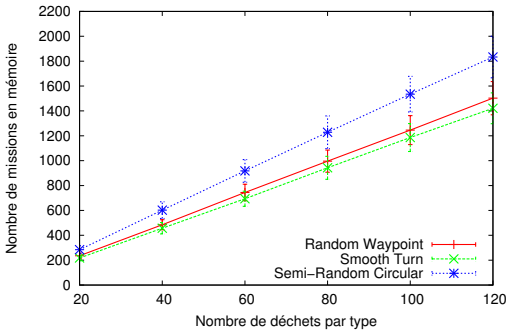
Nos résultats de simulation montrent une faible différence de performances entre les modèles de mobilité *Smooth Turn* et *Randow Waypoint*. Le *Smooth Turn* étant une version dérivée du *Random Walk*, ces résultats sont cohérents avec les précédentes études qui portent sur les différences de performances réseaux (*e.g.* débit, délai) entre ces deux modèles [147] (*c.f.* chapitre 3). La très légère avance du *Smooth Turn* est due au fait que ce modèle confère une meilleure uniformité dans la distribution des drones tout au long du scénario, ce qui implique une meilleure connectivité entre les nœuds et donc une meilleure collaboration grâce à des échanges de messages plus réguliers. Comme nous l'avions suggéré dans le paragraphe précédent, les performances du modèle *Semi-Random Circular* sont en léger retrait. La principale raison est que ce modèle impose aux drones d'effectuer de grandes distances (*i.e.* cercle complet) avant de revenir au point d'origine. Les robots terrestres étant statiques tant qu'ils n'ont pas de mission à effectuer, les communications entre les drones et les robots sont moins fréquentes. La connectivité de la flotte est donc un paramètre determinant quant à l'efficacité du système AMiRALE. Néanmoins, celui-ci reste fonctionnel même dans le cas d'une connectivité faible, rare et sporadique.



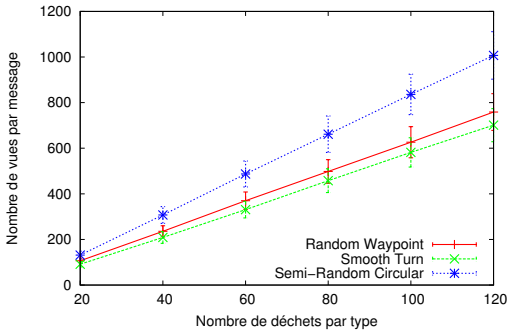
(a) *Temps total de nettoyage*



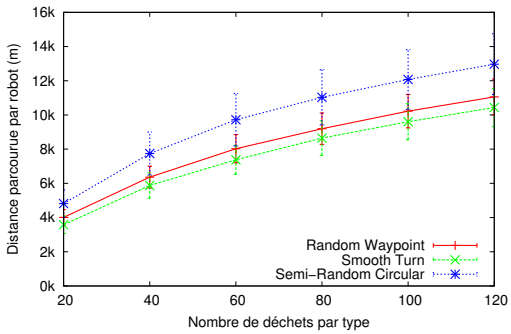
(b) *Nombre de missions annulées*



(c) *Nombre de missions en mémoire*



(d) *Nombre de vues par message*



(e) *Distance parcourue par robot*

FIGURE 9.22 – Résultats de simulation de performance pour AMiRALE en temps absolu en fonction du modèle de mobilité des drones.

Conclusion

Dans cette section, nous avons pu constater que le modèle de mobilité utilisé par la flotte (ou un sous-ensemble des engins) a un impact notable sur les performances de notre système AMiRALE. En effet, la connectivité induite par le modèle de mobilité utilisé a des répercussions sur l'ensemble des mesures effectuées : temps de nettoyage, nombre de missions annulées, nombre de missions générées et nombre de vues par message. Une meilleur connectivité confère donc à notre système de meilleurs performances. Néanmoins, AMiRALE reste fonctionnel même dans le cas d'une connectivité fortement limitée.

9.5 Démonstrateur ParCS-S2

Dans le cadre de cette étude, nous avons entrepris une collaboration avec la société *Mugen*[1] afin de réaliser un démonstrateur pour notre scénario de nettoyage de parc exploitant le système AMiRALE. Mugen est une start-up française qui développe un solution logicielle appelée *M2UV*, permettant à un ensemble d'utilisateurs d'interagir avec des objets mobiles communicants depuis un ensemble de terminaux mobiles (*e.g.* smartphone, tablette). M2UV fait le lien entre les objets et les utilisateurs (chacun associé à un terminal) par l'intermédiaire d'un serveur central. Ce dernier fournit une interface spécifique à chaque utilisateur en fonction des droits dont il dispose sur chacun des objets de la flotte. L'application permet également de transférer ou de partager ces droits entre les utilisateurs du système. Ainsi, il est possible de définir des rôles pour chacun des utilisateurs en rapport avec chacun des objets de la flotte, lui conférant ainsi un ensemble de droits spécifiques sur ces objets. La figure 9.23 illustre l'architecture générale de la solution M2UV de Mugen.

Le projet ParCS-S2 (*Park Cleaning Swarm Supervision System*) a pour objectif de combiner notre scénario de nettoyage de parc avec l'application de supervision M2UV. À l'heure actuelle, nous avons réalisé un démonstrateur virtuel reposant sur le simulateur JbotSim. Nous avons adapté son fonctionnement pour permettre aux nœuds mobiles de simuler le comportement de systèmes autonomes communicants réels et ainsi interagir avec l'application M2UV. Une illustration de ce démonstrateur est fournit à la figure 9.24.

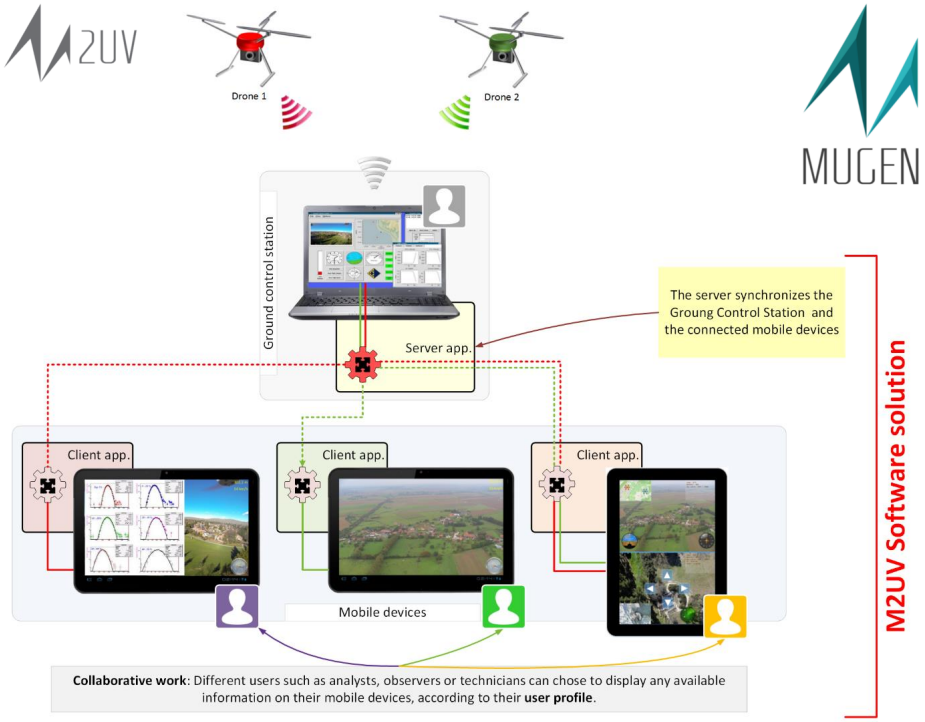


FIGURE 9.23 – Architecture de la solution M2UV de la start-up Mugen.

1. http://mugen-sas.com

FIGURE 9.24 – Illustration du démonstrateur ParCS-S2.

9.6 Résumé

Dans ce chapitre, nous avons évalué notre système collaboratif AMiRALE sous plusieurs angles en nous reposant sur un scénario d'application créé à cet effet. Le scénario ParCS permet à une flotte de systèmes autonomes spécialisés (*i.e.* robots et drones) de nettoyer collaborativement un ensemble de déchets de différentes natures présents sur le sol d'un parc. Nous avons effectué un certain nombre d'expérimentations par simulation ainsi que par émulation afin de démontrer les apports de notre système collaboratif distribué reposant uniquement sur des opérations locales et des communications asynchrones. Nous avons ainsi montré que notre modèle permet d'obtenir des résultats proches de ceux d'un modèle collaboratif centralisé (*i.e.* le Blackboard). Nous avons également montré l'intérêt des filtres de notre modèle, notamment dans le but de réduire la bande passante utilisée. Nous avons ainsi calculé et mesuré la quantité mémoire ainsi que la bande passante nécessaires à l'utilisation de notre système dans le cadre de ce scénario. De plus, nous avons mis en évidence le fait que les trois versions temporelles d'AMiRALE permettent d'obtenir des résultats similaires. En outre, nous avons montré que l'usage de la règle de préemption ne permet pas d'améliorer significativement l'efficacité de notre système dans le cadre de ce scénario. Enfin, nous avons montré qu'un modèle de mobilité possédant des propriétés de connectivité forte permet d'améliorer l'efficacité générale de notre système collaboratif.

Grâce à l'ensemble de ces expérimentations, nous pouvons conclure que :

— AMiRALE est résiliant aux pannes même si celles-ci se produisent à des fréquences élevées ;
— la quantité mémoire et la bande passante requises par AMiRALE sont compatibles avec celles fournies par les systèmes autonomes actuels ;
— AMiRALE est fonctionnel dans des flottes fortement déconnectées où les communications sont rares et sporadiques ;
— l'utilisation des filtres du modèle permet d'adapter et d'optimiser le comportement et les performances d'AMiRALE pour l'application visée.

Conclusion

Les systèmes autonomes représentent un domaine d'activité et de recherche en plein essor. L'objectif de cette thèse était de proposer un nouveau mécanisme de découverte de services permettant la collaboration entre les engins d'une flotte autonome fortement dynamique (*i.e.* mobile et volatile).

Contributions

État de l'art

Afin de proposer une solution nouvelle et pertinente, nous avons élaboré un état de l'art portant sur les différents aspects du problème.

Dans le chapitre 1, nous avons présenté les différents types de systèmes autonomes existants ainsi que les applications associées. Nous avons caractérisé les différents niveaux d'autonomie pour ces systèmes et avons précisé la définition de flotte autonome. Nous avons également détaillé l'ensemble des modèles d'interaction possibles dans ce type de flotte. Nous avons ainsi pu définir la notion de collaboration qui permet à un ensemble d'objets communicants de résoudre un problème de haut niveau en le subdivisant en tâches individuelles.

Dans le chapitre 2, après avoir discuté des différentes architectures de réseaux et des diverses technologies de communication utilisées au sein des flottes autonomes, nous avons pu déterminer que l'inondation était la méthode de diffusion la plus efficace dans un réseau soumis à une forte volatilité et à une forte mobilité. Ces deux facteurs contextuels ont en effet un impact déterminant dans la connectivité du réseau qui influe elle même de façon significative sur les performances des protocoles utilisés au sein d'une flotte.

C'est pourquoi dans le chapitre 3, nous avons présenté les modèles de mobilité les plus couramment utilisés pour les systèmes autonomes.

Dans le chapitre 4, nous avons précisé le concept de découverte de services qui permet aux nœuds d'un réseau de partager leurs capacités respectives. Un tel mécanisme est supposé fournir un ensemble de fonctionnalités telles que la localisation

d'un service, son invocation ou encore la sélection du fournisseur le plus pertinent. Néanmoins, la majorité des mécanismes de découverte de services existants ne fournissent que la fonction de localisation, délaissant ainsi les fonctions de sélection et d'usage des services découverts. Nous avons également pu mettre en évidence que le contexte (*e.g.* la mobilité des nœuds, la taille du réseau) a un impact significatif sur les performances de ce type de système. Nous avons en outre pu déterminer que, dans le cas des réseaux fortement dynamiques, le modèle sans annuaire reposant sur une stratégie d'inondation contrôlée semble être le plus efficace.

Nous avons enfin montré dans le chapitre 5 que les solutions de découverte de services existantes ne répondaient pas aux caractéristiques propres aux flottes autonomes collaboratives que nous avons présentées dans les chapitres 1 et 2.

AMiRALE

Fort de cet état de l'art, nous avons proposé dans la partie II de cette thèse un nouveau système appelé AMiRALE qui permet aux engins mobiles d'une flotte hétérogène fortement dynamique de collaborer. Notre système est exclusivement basé sur des opérations locales et des communications asynchrones.

Le chapitre 6 décrit le fonctionnement général du système ainsi que ses caractéristiques. Nous avons introduit un mécanisme appelé filtre permettant d'adapter le comportement de certains éléments qui composent notre système afin de prendre compte les spécificités de l'application qui l'utilise. Nous avons également pris en compte le phénomène de dérive des horloges en proposant une version spécifique de notre système par stratégie de synchronisation temporelle.

Le chapitre 7 présente une formalisation complète d'AMiRALE basée sur des règles de ré-étiquetage de graphes dynamiques.

NEmu

Dans le chapitre 8, nous avons présenté un nouvel outil de virtualisation de réseaux mobiles appelé NEmu. Celui-ci permet d'instancier des réseaux virtuels statiques, dynamiques ou mobiles avec un contrôle important sur les propriétés de la topologie de ce réseau ainsi que sur la configuration des nœuds et des inter-connexions. NEmu est notamment utilisé dans le chapitre 9 comme plate-forme d'expérimentation pour l'évaluation de notre système AMiRALE.

ParCS

Enfin, nous avons conçu le scénario d'exécution ParCS dans le but d'évaluer notre système collaboratif. Ce scénario permet à une flotte autonome de drones et de robots terrestres de nettoyer l'ensemble des déchets présents sur le sol d'un parc. Chaque robot terrestre étant spécialisé dans le traitement d'un seul type de déchet, AMiRALE est ici utilisé pour permettre aux différents engins de la flotte de collaborer afin de réduire notamment le temps nécessaire au nettoyage de l'intégralité du parc.

Nous avons ainsi pu montrer dans le chapitre 9 que notre système AMiRALE permet d'obtenir un temps de nettoyage proche de celui d'un mécanisme collaboratif centralisé. Au travers des différentes expérimentations présentées dans ce chapitre, effectuées par simulation ou par émulation, nous avons pu démontrer que les différentes versions de notre mécanisme AMiRALE sont tolérantes au pannes des engins même si celles-ci interviennent à une fréquence élevée. Nous avons également pu montrer que la quantité de mémoire ainsi que la bande passante nécessaires au fonctionnement de notre système sont compatibles avec celles proposées par les systèmes autonomes actuels.

Nous avons enfin présenté un démonstrateur virtuel de notre scénario de nettoyage de parc réalisé en collaboration avec la société Mugen.

Travaux futurs

À l'avenir, nous aimerions exploiter la formalisation du système AMiRALE afin d'identifier des propriétés telles que le temps de convergence ou encore le nombre optimal d'entités en fonction des paramètres du scénario. Cette étude nous permettrait de proposer une méthodologie afin de déterminer les valeurs des paramètres internes de notre système. En outre, nous souhaiterions étendre AMiRALE afin qu'il puisse intégrer des comportements en coalition permettant ainsi au système de gérér des tâches complexes qui nécessitent le recours à plusieurs entités.

Nous souhaitons également réaliser un démonstrateur physique de notre scénario de nettoyage de parc afin de démontrer son fonctionnement en condition réelle.

Enfin, nous souhaitons continuer le développement de NEmu en lui ajoutant notamment des fonctionnalités de réseaux programmables [260].

Annexes

ANNEXE A

Publications

A.1 Conférences

Distributed Collaborative System for Heterogeneous Swarms of Autonomous Mobile Robots
Vincent Autefage, Serge Chaumette et Damien Magoni
Ingénierie des Protocoles et Nouvelles Technologies de la Répartition (IEEE CFIP 16th / NOTERE 12th)
Paris, France, Juillet 2015

A Mission-Oriented Service Discovery Mechanism for Highly Dynamic Autonomous Swarms of Unmanned Systems
Vincent Autefage, Serge Chaumette et Damien Magoni
International Conference on Autonomic Computing (IEEE ICAC 12th)
Grenoble, France, Juillet 2015

Comparison of Time Synchronization Techniques in a Distributed Collaborative Swarm System
Vincent Autefage, Serge Chaumette et Damien Magoni
European Conference on Networks and Communications (IEEE EuCNC 24th)
Paris, France, Juin/Juillet 2015

Influence des modèles de mobilité sur un système collaboratif pour flottes autonomes hétérogènes
Vincent Autefage, Serge Chaumette et Damien Magoni
Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (ALGOTEL 17th)
Beaune, France, Juin 2015

ParCS-S2 : Park Cleaning Swarm Supervision System - a position paper
Vincent Autefage, Arnaud Casteler, Serge Chaumette, Nicolas Daguisé, Arnaud Dutartre et Tristan Mehamli
International Aerospace Supply Fair Congress (AIRTEC 9th)
Francfort, Allemagne, Octobre 2014

NEmu : un outil de virtualisation de réseaux à la demande pour l'enseignement
Vincent Autefage et Damien Magoni
Journées RESeaux de l'Enseignement et de la Recherche (RENATER JRES 10th)
Montpellier, France, Décembre 2013

Virtualisation de réseau avec Network Emulator et application à l'évaluation d'un réseau recouvrant
Vincent Autefage et Damien Magoni
Nouvelles Technologies de la Répartition et Ingénierie des Protocoles (NOTERE 11th / CFIP 15th)
Best paper award
Anglet, France, Octobre 2012

Network Emulator : a Network Virtualization Testbed for Overlay Experimentations
Vincent Autefage et Damien Magoni
International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (IEEE CAMAD 17th)
Barcelone, Espagne, Septembre 2012

A.2 Journal

Virtualisation distribuée de réseaux dynamiques et mobiles avec NEmu
Vincent Autefage et Damien Magoni
HERMANN RNTI-SM-2
Mai 2013

A.3 Rapports techniques

AMiRALE Formal Model - A Service Discovery and Collaboration System Formalism based on Dynamic Graph Relabeling
Vincent Autefage
Université de Bordeaux - LaBRI
Février 2015

ArDroneXT - Ar.Drone 2 eXTension for swarming and service hosting
Vincent Autefage et Serge Chaumette
Université de Bordeaux - LaBRI
Décembre 2013

ANNEXE B

Logiciels

B.1 AMiRALE

AMiRALE (*Asynchronous Missions Relay for Autonomous and Lively Entities*) est une implémentation complète de notre système collaboratif de découverte de services. Elle est distribuée sous la forme d'une bibliothèque JAVA sans dépendance externe. Cette implémentation est utilisée pour obtenir l'ensemble des résultats présentés dans cette thèse.

B.2 ParCS-S2

ParCS-S2 (*Park Cleaning Swarm Supervision System*) est le résultat de la collaboration entre le LaBRI et la société Mugen. Ce démonstrateur se base sur notre scénario de nettoyage de parc présenté dans la thèse. Il combine l'utilisation du simulateur de graphes dynamiques JBotSim et de la solution de contrôle M2UV développée par Mugen. Ce démonstrateur a fait l'objet d'une présentation publique lors de la convention *UAV/ADS Show Europe 2014*.

B.3 NEmu

NEmu (*Network Emulator for Mobile Universes*) est un environnement permettant d'instancier des réseaux virtuels statiques, dynamiques ou mobiles avec un contrôle important sur les propriétés de la topologie ainsi que sur la configuration des nœuds et des inter-connexions. Outre son utilisation dans le cadre de cette thèse, cet outil a fait l'objet de plusieurs publications. Il est actuellement utilisé dans de nombreux

cursus informatiques de l'université de Bordeaux (IUT, Licence professionnelle, Licence générale, Master 1, Master 2, Miage).

B.4 ArDroneXT

ArDroneXT (*Ar.Drone 2 eXTension*) est une surcharge du système d'exploitation du Parrot Ar.Drone 2. Ce système alternatif permet la création d'une flotte autonome d'Ar.Drone 2. Cette solution ajoute le support des communications directes entre ces appareils mais également de nouveaux services logiciels sur les drones. Couplé à une API développée au cours de cette thèse, ce système permet le contrôle de plusieurs engins par le même opérateur à l'aide d'interfaces variées (e.g. clavier, souris, joystick, kinect). Ce système est au cœur de la plupart des actions de médiation présentées plus loin dans ce document. ArDroneXT est utilisé au sein des enseignements du *Master 2 RSM* de l'université de Bordeaux. Une présentation courte de ce système a été faite lors de la conférence *JRES 2013*. Enfin, il a été utilisé dans un tutoriel sur le *swarming* lors de la conférence *AETOS* en 2014.

ANNEXE C

Actions de médiation scientifique

C.1 Publications

Un support d'acculturation à un domaine technologique
Vincent Autefage
1024 - Bulletin de la société informatique de France
Hors-série 1 - Médiation scientifique : de la science informatique au grand public
Février 2015

Un support d'acculturation à un domaine technologique - contrôle gestuel d'un essaim de drones
Vincent Autefage
Congrès de la société informatique de France 2014
Poitiers, France, Février 2014

C.2 Présentations et démonstrations

Nuit des chercheurs : Flottes de drones et de robots
Vincent Autefage
Cap Sciences
Bordeaux, Septembre 2015

Comment faire collaborer des drones et des robots ?
Vincent Autefage
Cap Sciences - Radio RCF
Bordeaux, Mai 2015

Comment utiliser des drones et des robots pour faire le ménage efficacement ?
Vincent Autefage
Université de Bordeaux - Ascoergo
Bordeaux, Avril 2015

Épistémologie, histoire et révolution numérique : Drones et Informatique
Vincent Autefage et Serge Chaumette
Université de Bordeaux - LaBRI
Bordeaux, Mars 2015

Maths à modeler : Introduction aux drones civils
Vincent Autefage
Université de Bordeaux - LaBRI
Bordeaux, Février 2015

Semaine Digitale - Nuit du Web : Rendez-vous avec les objets communicants
Vincent Autefage et Serge Chaumette
Ville de Bordeaux
Bordeaux, Octobre 2014

Fête de la science : Introduction aux drones civils
Vincent Autefage
Université de Bordeaux - LaBRI - Cap Sciences
Bordeaux, Octobre 2014

Nuit des chercheurs : Nettoyer un parc avec une flotte de drones
Vincent Autefage
Cap Sciences
Bordeaux, Septembre 2014

Lab 2 l'été : Contrôle gestuel d'une flotte de drones
Vincent Autefage
Cap Sciences
Bordeaux, Août 2014

Ma thèse en 180 secondes
Vincent Autefage
Université de Bordeaux - Région Aquitaine
Prix du public de la finale régionale
Bordeaux, Mai 2014

Sciences à l'antenne : Sous l'œil des drones
Vincent Autefage, Serge Chaumette et Pierre Melchior
Cap Sciences - Radio Campus Bordeaux
Bordeaux, Mai 2014

Conférence EMF : Introduction aux drones civils
Vincent Autefage
Société Informatique de France - Espace Mendès France
Poitiers, Février 2014

NOVAQT : Contrôle gestuel d'une flotte de drones
Vincent Autefage
Région Aquitaine - ADI - AeroCampus Aquitaine
Bordeaux, Octobre 2013

Fête de la science : Introduction aux drones civils
Vincent Autefage
Université de Bordeaux - LaBRI - Cap Sciences
Bordeaux, Octobre 2013

Nuit des chercheurs : Contrôle gestuel d'une flotte de drones
Vincent Autefage
Cap Sciences
Bordeaux, Septembre 2013

Lab de l'été : Contrôle gestuel d'un drone
Vincent Autefage
Cap Sciences
Bordeaux, Août 2013

Mardi des sciences : Introduction aux drones civils
Vincent Autefage
Cap Sciences
Bordeaux, Juin 2013

Let's talk about drones
Vincent Autefage
Université de Bordeaux
Bordeaux, Juin 2013

Bibliographie

[1] Instituts Carnot : Le livre blanc - objets communicants et internet des objets. Rapport technique, June 2011.

[2] Joseph Bradley, Joel Barbier et Doug Handler : L'internet of everything, livre blanc. Rapport technique, 2013. CISCO Systems.

[3] Cap-Digital : Cahier de tendances - marchés et leviers. Rapport technique, 2014.

[4] Antonio Regalado : Business adapts to a new style of computer. *MIT Technlology Review - The Internet of Things*, pages 1–2, July/August 2014.

[5] CERNA : Ethique de la recherche en robotique. Rapport technique, November 2014.

[6] République Française : France robots initiatives. Rapport technique, Ministère du redressement productif et Ministère de l'enseignement supérieur et de la recherche, March 2013.

[7] G. Dudek, M. Jenkin, E. Milios et D. Wilkes : A taxonomy for swarm robots. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems 93 (IROS 93)*, volume 1, pages 441–447, July 1993.

[8] NIST : Autonomy levels for unmanned systems. Rapport technique, Otcober 2008.

[9] European RPAS Steering Group : Roadmap for the integration of civil remotely-piloted aircraft systems into the european aviation system. Rapport technique, June 2013.

[10] General Atomics : *MQ-9 Reaper (Predator B RPA)*. http://www.ga-asi.com/predator-b.

[11] United States of America : Unmanned systems integrated roadmap. Rapport technique, Departement of Defense, 2013.

[12] République Française : *La DGA receptionne le systeme Reaper francais*. Direction Générale de l'Armement, 2014. http://www.defense.gouv.fr/actualites/economie-et-technologie/la-dga-receptionne-le-systeme-reaper-francais.

[13] Delair Tech : *DT-18*. http://www.delair-tech.com.

[14] Fly n Sense : *X4*. http://www.fly-n-sense.com.

[15] Parrot : *ArDrone 2.0*. http://ardrone2.parrot.com.

[16] Phodia : *Gélule Ultimate 2*. http://www.phodia.com.

[17] Festo : *BionicOpter - Inspired by dragonfly flight*, 2013. www.festo.com/bionics.

[18] Vitirover : *Vitirover - Micro Robotique Viticole*. http://www.vitirover.com.

[19] R&DTech France : *ROBCO S - Drone de reconnaissance*. http://www.retdtechfrance.fr.

[20] Orbotix : *Sphero*. http://www.gosphero.com/fr/ollie.

[21] Parrot : *Jumping Sumo*. http://www.parrot.com/fr/produits/jumping-sumo.

[22] R&DTech France : *SKOPY 250 - Robot Terrestre*. http://www.retdtechfrance.fr.

[23] SDR Tactical Robots : *LT2/F "Bulldog" - Light Tracked Robot with Arm*. http://www.sdrtactical.com/LT2Arm-Bulldog.

[24] iRobot : *SUGV 310*. http://www.irobot.fr/a-propos-d-irobot/defense-securite.

[25] Boston Dynamics : *BigDog - The Most Advanced Rough-Terrain Robot on Earth*. http://www.bostondynamics.com/robot_bigdog.html.

[26] Boston Dynamics : *RiSE : The Amazing Climbing Robot*. http://www.bostondynamics.com/robot_rise.html.

[27] Sony : *Aibo*. http://www.sony-aibo.com.

[28] Aldebaran : *NAO*. https://www.aldebaran.com/en/humanoid-robot/nao-robot.

[29] Deepak Trivedi, Christopher D Rahn, William M Kier et Ian D Walker : Soft robotics : Biological inspiration, state of the art, and future research. *Applied Bionics and Biomechanics*, 5(3):99–117, 2008.

[30] Biorobotics et Biomechanics Lab : *Snake Robot*. http://brml.technion.ac.il.

[31] Sangok Seok, C.D. Onal, Kyu-Jin Cho, R.J. Wood, D. Rus et Sangbae Kim : Meshworm : A peristaltic soft robot with antagonistic nickel titanium coil actuators. *IEEE/ASME Transactions on Mechatronics*, 18(5):1485–1497, October 2013.

[32] Orbotix : *Sphero*. http://www.gosphero.com/fr/sphero.

[33] M. Rubenstein, C. Ahler et R. Nagpal : Kilobot : A low cost scalable robot system for collective behaviors. *In IEEE International Conference on Robotics and Automation 2012 (ICRA 2012)*, pages 3293–3298, May 2012.

[34] Amit Motwani : A survey of uninhabited surface vehicless. Rapport technique, April 2002.

[35] Thomas B. Curtin Robert W. Button, John Kamp et James Dryden : A survey of missions for unmanned undersea vehicles. Rapport technique, 2009.

[36] Qualcomm : *Snapdragon Cargo*. https://www.qualcomm.com/invention/research/projects/robotics.

[37] Parrot : *Rolling Spider*. http://www.parrot.com/fr/produits/rolling-spider.

[38] République Française : *Arrêté relatif à la conception des aéronefs civils qui circulent sans aucune personne à bord, aux conditions de leur emploi et sur les capacités requises des personnes qui les utilisent*. Ministère de l'écologie, du développement durable et de l'énergie - Direction générale de l'aviation civile, April 2012.

[39] République Française : *Arrêté relatif à l'utilisation de l'espace aérien par les aéronefs qui circulent sans personne à bord*. Ministère de l'écologie, du développement durable et de l'énergie - Direction générale de l'aviation civile, April 2012.

[40] République Française : *Drones (aéronefs télépilotés)*. Ministère de l'écologie, du développement durable et de l'énergie - Direction générale de l'aviation civile, 2015. http://www.developpement-durable.gouv.fr/-Drones-aeronefs-telepilotes-.html.

[41] SenseFly : *eBee*. https://www.sensefly.com/drones/ebee.html.

[42] J.O. Kephart et D.M. Chess : The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, January 2003.

[43] MathWorks : *MATLAB - Le langage du calcul scientifique*. http://fr.mathworks.com/products/matlab.

[44] Open Source Robotics Foundation : *The Robot Operating System (ROS)*. http://www.ros.org.

[45] M. Turpin, N. Michael et V. Kumar : Decentralized formation control with variable shapes for aerial robots. *In IEEE International Conference on Robotics and Automation 2012 (ICRA 2012)*, pages 23–30, May 2012.

[46] Alex Kushleyev, Daniel Mellinger, Caitlin Powers et Vijay Kumar : Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300, 2013.

[47] J.P. How, B. Bethke, A. Frank, D. Dale et J. Vian : Real-time indoor autonomous vehicle test environment. *IEEE Control Systems*, 28(2):51–64, April 2008.

[48] Sergei Lupashin, Markus Hehn, Mark W. Mueller, Angela P. Schoellig, Michael Sherback et Raffaello D'Andrea : A platform for aerial robotics research and demonstration : The flying machine arena. *Mechatronics*, 24(1):41–54, 2014.

[49] Y.U. Cao, A.S. Fukunaga, A.B. Kahng et F. Meng : Cooperative mobile robotics : antecedents and directions. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems 95, 'Human Robot Interaction and Cooperative Robots'*, volume 1, pages 226–234, August 1995.

[50] Y.Uny Cao, AlexS. Fukunaga et Andrew Kahng : Cooperative mobile robotics : Antecedents and directions. *Autonomous Robots*, 4(1):7–27, 1997.

[51] Luca Iocchi, Daniele Nardi et Massimiliano Salerno : Reactivity and deliberation : a survey on multi-robot systems. *In Balancing reactivity and social deliberation in multi-agent systems*, pages 9–32. Springer, 2001.

[52] Lynne E Parker : Multiple mobile robot systems. *In Springer Handbook of Robotics*, pages 921–941. Springer, 2008.

[53] Y. Mohan et S.G. Ponnambalam : An extensive review of research in swarm robotics. *In World Congress on Nature Biologically Inspired Computing 2009 (NaBIC 2009)*, pages 140–145, December 2009.

[54] Nicole El Zoghby, Valeria Loscri, Enrico Natalizio et Véronique Cherfaoui : Robot cooperation and swarm intelligence. *Wireless Sensor and Robot Networks : From Topology Control to Communication Aspects*, 2014.

[55] Lynne E Parker : Reliability and fault tolerance in collective robot systems. *Handbook on Collective Robotics : Fundamentals and Challenges*, 2012. Pan Stanford Publishing.

[56] KMel Robotics : *Meet Your Creator - Quadrotor Show*, 2012. https://www.grasp.upenn.edu/success_story/meet_your_creator_quadrotor_show.

[57] Puy du fou : *Le Puy du Fou dans la bataille mondiale des drones*. http://www.puydufou.com/histoires-de-ouf/le-puy-du-fou-dans-la-bataille-mondiale-des-drones.html.

[58] James Alexander Stark, Clifford Wong et Robert Scott Trowbridge : *Controlling unmanned aerial vehicles as a flock to synchronize flight in aerial displays*. Disney Enterprises Inc., September 2014. Brevet numéro 20140249693.

[59] Kurt Konolige, Dieter Fox, Charlie Ortiz, Andrew Agno, Michael Eriksen, Benson Limketkai, Jonathan Ko, Benoit Morisset, Dirk Schulz, Benjamin Stewart et Regis Vincent : Centibots : Very large scale distributed robotic teams. *In Experimental Robotics IX*, volume 21 de *Springer Tracts in Advanced Robotics*, pages 131–140. Springer Berlin Heidelberg, 2006.

[60] Regis Vincent, Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Benoit Morisset, Charles Ortiz, Dirk Schulz et Benjamin Stewart : Distributed multirobot exploration, mapping, and task allocation. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):229–255, 2008.

[61] Michael Rubenstein, Alejandro Cornejo et Radhika Nagpal : Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[62] Nathan Michael, Shaojie Shen, Kartik Mohta, Yash Mulgaonkar, Vijay Kumar, Keiji Nagatani, Yoshito Okada, Seiga Kiribayashi, Kazuki Otake, Kazuya Yoshida, Kazunori Ohno, Eijiro Takeuchi et Satoshi Tadokoro : Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, September 2012.

[63] Robert Grabowski, Luis Ernesto Navarro-Serment, Chris Paredis et Pradeep Khosla : Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots - Special Issue on Heterogeneous Multirobot Systems*, 1999.

[64] Alessandro Stranieri, Eliseo Ferrante, Ali Emre Turgut, Vito Trianni, Carlo Pinciroli, Mauro Birattari et Marco Dorigo : Self-organized flocking with a heterogeneous mobile robot swarm. *In Advances in Artificial Life (ECAL)*, pages 789–796. MIT press, August 2011.

[65] Gregory Dudek, MichaelR.M. Jenkin, Evangelos Milios et David Wilkes : A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397, 1996.

[66] A. Farinelli, L. Iocchi et D. Nardi : Multirobot systems : a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B : Cybernetics*, 34(5):2015–2028, October 2004.

[67] Lynne E Parker : Distributed intelligence : Overview of the field and its application in multi-robot systems. *In AAAI Fall Symposium on Regarding the Intelligence in Distributed Intelligent Systems*, 2007.

[68] Lynne E Parker : Distributed intelligence : Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, 2(1):5–14, 2008.

[69] E. Kuiper et S. Nadjm-Tehrani : Mobility models for uav group reconnaissance applications. *In International Conference on Wireless and Mobile Communications 2006 (ICWMC '06)*, July 2006.

[70] C.Ronald Kube et Eric Bonabeau : Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1–2):85–101, 2000.

[71] Fabrice R. Noreils : Toward a robot architecture integrating cooperation between mobile robots : Application to indoor environment. *The International Journal of Robotics Research*, 12(1):79–98, 1993.

[72] S. Chaumette, R. Laplace, C. Mazel, R. Mirault, A. Dunand, Y. Lecoutre et J.-N. Perbet : Carus, an operational retasking application for a swarm of auto-

nomous uavs : First return on experience. *In MILCOM*, pages 2003–2010, November 2011.

[73] Daniel D Corkill : Blackboard systems. *AI expert*, 6(9):40–47, 1991.

[74] Daniel D. Corkill : Design alternatives for parallel and distributed blackboard systems. Rapport technique, 1988.

[75] L.E. Parker et Fang Tang : Building multirobot coalitions through automated task solution synthesis. *Proceedings of the IEEE*, 94(7):1289–1305, July 2006.

[76] L. Vig et J.A. Adams : Multi-robot coalition formation. *IEEE Transactions on Robotics*, 22(4):637–649, August 2006.

[77] H. Asama, K. Ozaki, H. Itakura, A. Matsumoto, Y. Ishida et I. Endo : Collision avoidance among multiple mobile robots based on rules and communication. *In Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems '91 (IROS '91), 'Intelligence for Mechanical Systems'*, volume 3, pages 1215–1220, November 1991.

[78] M. Peasgood, C.M. Clark et J. McPhee : A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*, 24(2):283–292, April 2008.

[79] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawai et Hitoshi Matsubara : Robocup : A challenge problem for ai. *AI Magazine*, 18(1):73–85, 1997.

[80] The Robocup Federation : *RoboCup*. http://www.robocup.org.

[81] Jungfang Wang, Bin Xie et DharmaP. Agrawal : Journey from mobile ad hoc networks to wireless mesh networks. *In Guide to Wireless Mesh Networks*, Computer Communications and Networks, pages 1–30. Springer, 2009.

[82] J Mackar et S Corson : *Mobile Ad hoc Networking (MANET) : Routing Protocol Performance Issues and Evaluation Considerations*. IETF, January 1999. rfc 2501.

[83] Andrew S. Tanenbaum et David J. Wetherall : *Computer Networks*. Pearson, 5th édition, 2011.

[84] Mehmet S. Kuran et Tuna Tugcu : A survey on emerging broadband wireless access technologies. *Computer Networks*, 51(11):3013–3046, 2007.

[85] IEEE : *802.15.1 : Wireless Personnal Area Networks (WPANs)*, 2005.

[86] IEEE : *802.15.4 : Low-Rate Wireless Personal Area Networks (LR-WPANs)*, 2011.

[87] C. Gomez et J. Paradells : Wireless home automation networks : A survey of architectures and technologies. *IEEE Communications Magazine*, 48(6):92–101, June 2010.

[88] S. Chaumette, R. Laplace, C. Mazel et R. Mirault : Scual, swarm of communicating uavs at labri : An open uavnet testbed. *In International Symposium on Wireless Personal Multimedia Communications (WPMC 14th)*, pages 1–5, October 2011.

[89] ECMA : *Standard ECMA-340 : Near Field Communication - Interface and Protocol (NFCIP-1)*, 2013.

[90] ECMA : *Standard ECMA-352 : Near Field Communication - Interface and Protocol (NFCIP-2)*, 2013.

[91] J.M. Kahn et J.R. Barry : Wireless infrared communications. *Proceedings of the IEEE*, 85(2):265–298, February 1997.

[92] Jeffrey B. Carruthers : *Wireless Infrared Communications*. 2003.

[93] S. Kornienko, O. Kornienko et P. Levi : Minimalistic approach towards communication and perception in microrobotic swarms. *In International Conference on IEEE/RSJ Intelligent Robots and Systems 2005 (IROS 2005)*, pages 2228–2234, August 2005.

[94] Sergey Kornienko et Olga Kornienko : Ir-based communication and perception in microrobotic swarms. *CoRR*, abs/1109.3617, 2011.

[95] IEEE : *802.11 : Wireless Local Area Networks (WLANs)*, 2012.

[96] Parrot : *Bebop Drone*. http://www.parrot.com/fr/produits/bebop-drone.

[97] C. Perera, C.H. Liu et S. Jayawardena : The emerging internet of things marketplace from an industrial perspective : A survey. *IEEE Transactions on Emerging Topics in Computing*, (99):1–13, 2015.

[98] IEEE : *802.16 : Broadband Wireless Metropolitan Area Networks (WMANs)*, 2012.

[99] Pulkit Gupta : Evolvement of mobile generations : 1g to 5g. *International Journal For Technological Research In Engineering*, 1(3):152–157, June 2013.

[100] IEEE : *802.22 : Wireless Regional Area Networks*, 2011.

[101] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran et Shantidev Mohanty : Next generation/dynamic spectrum access/cognitive radio wireless networks : A survey. *Computer Networks*, 50(13):2127–2159, 2006.

[102] Google : *Project Loon*. http://www.google.com/loon.

[103] Facebook : *internet.org*. https://www.internet.org.

[104] C.J. Hegarty et E. Chatre : Evolution of the global navigation satellitesystem (gnss). *Proceedings of the IEEE*, 96(12):1902–1917, December 2008.

[105] R. Cavallari, F. Martelli, R. Rosini, C. Buratti et R. Verdone : A survey on wireless body area networks : Technologies and design challenges. *IEEE Communications Surveys Tutorials*, 16(3):1635–1657, Third 2014.

[106] S. Movassaghi, M. Abolhasan, J. Lipman, D. Smith et A. Jamalipour : Wireless body area networks : A survey. *IEEE Communications Surveys Tutorials*, 16(3):1658–1686, Third 2014.

[107] C. Maihofer : A survey of geocast routing protocols. *IEEE Communications Surveys Tutorials*, 6(2):32–42, 2004.

[108] Mehran Abolhasan, Tadeusz Wysocki et Eryk Dutkiewicz : A review of routing protocols for mobile ad hoc networks. *Ad hoc networks*, 2(1):1–22, 2004.

[109] Anuj K Gupta, Harsh Sadawarti et Anil K Verma : Review of various routing protocols for manets. *International Journal of Information and Electronics Engineering*, 1(3):251–259, 2011.

[110] Jean Carle, Nathalie Mitton, Olivier Flauzac, Bachar Salim Haggar et Florent Nolot : État de l'art sur les protocoles de routage dans les réseaux ad hoc. Rapport technique, RISC : Réseaux hétérogènes intelligents pour Situation de Crise, July 2008.

[111] Nadjib Badache, D Djenour, Abdelouahid Derhab et Tayeb Lemlouma : Les protocoles de routage dans les réseaux mobiles ad hoc. *Revue d'Information Scientifique et Technique*, 12(2):77–112, 2004.

[112] Charles E Perkins et Pravin Bhagwat : Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *In ACM SIGCOMM Computer Communication Review*, volume 24, pages 234–244. ACM, 1994.

[113] Tsu-Wei Chen et M. Gerla : Global state routing : a new routing scheme for ad-hoc wireless networks. *In Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on*, volume 1, pages 171–175, Jun 1998.

[114] A. Iwata, Ching-Chuan Chiang, Guangyu Pei, M. Gerla et Tsu-Wei Chen : Scalable routing strategies for ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1369–1379, August 1999.

[115] T Clausen et P Jaqcquet : *Optimized Link State Routing Protocol (OLSR)*, October 2003. RFC 3626.

[116] Ulrich Herberg, Thomas Clausen, Philippe Jacquet et Christopher Dearlove : *The optimized link state routing protocol version 2*, April 2014. RFC 7181.

[117] David B Johnson et David A Maltz : Dynamic source routing in ad hoc wireless networks. *In Mobile computing*, pages 153–181. Springer, 1996.

[118] C.E. Perkins et E.M. Royer : Ad-hoc on-demand distance vector routing. *In Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications 1999 (WMCSA '99.)*, pages 90–100, February 1999.

[119] Charles Perkins, E Belding-Royer et Samir Das : *Ad hoc on demand distance vector (AODV) routing*, July 2003. RFC 3561.

[120] Vincent D. Park et M. Scott Corson : A highly adaptive distributed routing algorithm for mobile wireless networks. *In Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Driving the Information Revolution (INFOCOM '97)*, pages 1405–1414, 1997.

[121] ZygmuntJ. Haas et MarcR. Pearlman : Evaluation of the ad-hoc connectivity with the zone routing protocols. *In Wireless Personal Communications*, volume 482 de *The International Series in Engineering and Computer Science*, pages 201–212. Springer, 2002.

[122] Z.J. Haas : A new routing protocol for the reconfigurable wireless networks. *In IEEE 6th International Conference on Universal Personal Communications Record*, volume 2, pages 562–566, October 1997.

[123] M. Joa-Ng et I-Tai Lu : A peer-to-peer zone-based two-level link state routing for mobile ad hoc networks. *Selected Areas in Communications, IEEE Journal on*, 17(8):1415–1425, August 1999.

[124] I. Stojmenovic : Position-based routing in ad hoc networks. *IEEE Communications Magazine*, 40(7):128–134, July 2002.

[125] C. Lemmon, Siu Man Lui et Ickjai Lee : Geographic forwarding and routing for ad-hoc wireless network : A survey. *In Fifth International Joint Conference on INC, IMS and IDC, 2009 (NCM '09.)*, pages 188–195, August 2009.

[126] A. Khelil, C. Becker, J. Tian et K. Rothermel : An epidemic model for information diffusion in manets. *In Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 54–60. ACM, 2002.

[127] S.Y. Ni, Y.C. Tseng, Y.S. Chen et J.P. Sheu : The broadcast storm problem in a mobile ad hoc network. *In Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162. ACM, 1999.

[128] Sunil Kumar, Vineet S. Raghavan et Jing Deng : Medium access control protocols for ad hoc wireless networks : A survey. *Ad Hoc Networks*, 4(3):326–358, 2006.

[129] Brad Williams et Tracy Camp : Comparison of broadcasting techniques for mobile ad hoc networks. *In Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking &Amp ; Computing (MobiHoc '02)*, pages 194–205. ACM, 2002.

[130] Thomas Clausen, Brian Adamson et Christopher Dearlove : *Jitter considerations in mobile ad hoc networks (MANETs)*. IETF, February 2008. rfc 5148.

[131] J.A. Cordero, P. Jacquet et E. Baccelli : Impact of jitter-based techniques on flooding over wireless ad hoc networks : Model and analysis. *In Proceedings of IEEE INFOCOM*, pages 2059–2067, March 2012.

[132] Ashish Jain, Marco Gruteser, Mike Neufeld et Dirk Grunwald : Benefits of packet aggregation in ad-hoc wireless network. Rapport technique, University of Colorado, August 2003. CU-CS-960-03.

[133] D. E. Cooper, P. Ezhilchelvan et I. Mitrani : High coverage broadcasting for mobile ad-hoc networks. *In The Third IFIPTC6 Networking Conference*, pages 100–111, 2004.

[134] Y. Sasson, D. Cavin et A. Schiper : Probabilistic broadcast for flooding in wireless mobile ad hoc networks. *In IEEE Wireless Communications and Networking 2003 (WCNC 2003)*, volume 2, pages 1124–1130, March 2003.

[135] Patricia Ruiz et Pascal Bouvry : On the improvement of the enhanced distance-based broadcasting algorithm. *International Journal of Communication Networks and Distributed Systems*, 9(3/4):225–246, August 2012.

[136] L. Hogie, P. Bouvry, M. Seredynski et F. Guinand : A bandwidth-efficient broadcasting protocol for mobile multi-hop ad hoc networks. *In International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, pages 1–9, April 2006.

[137] Luc Hogie, Grégoire Danoy, Pascal Bouvry et Frédéric Guinand : A context-aware broadcast protocol for mobile wireless networks. *In Modelling, Computation and Optimization in Information Systems and Management Sciences*, volume 14 de *Communications in Computer and Information Science*, pages 507–519. Springer, 2008.

[138] C. Mbarushimana et A. Shahrabi : Comparative study of reactive and proactive routing protocols performance in mobile ad hoc networks. *In 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW '07)*, volume 2, pages 679–684, May 2007.

[139] N.I. Sarkar et W.G. Lol : A study of manet routing protocols : Joint node density, packet length and mobility. *In IEEE Symposium on Computers and Communications 2010 (ISCC 2010)*, pages 515–520, June 2010.

[140] Salima Hamma, Eddy Cizeron, Hafiz Issaka et V. Guédon, Jean-Pierre : Performance evaluation of reactive and proactive routing protocol in IEEE 802.11 ad hoc network. *In ITCom 06 - next generation and sensor networks*, October 2006.

[141] V. Del Duca Almeida, A.B. Oliveira, D.F. Macedo et J.M.S. Nogueira : Performance evaluation of manet and dtn routing protocols. *In IFIP Wireless Days 2012 (WD 2012)*, pages 1–6, November 2012.

[142] E.M. Royer, P.M. Melliar-Smith et L.E. Moser : An analysis of the optimum node density for ad hoc mobile networks. *In IEEE International Conference on Communications 2001 (ICC 2001)*, volume 3, pages 857–861, 2001.

[143] D. Nguyen et P. Minet : Scalability of the olsr protocol with the fish eye extension. *In Sixth International Conference on Networking 2007 (ICN '07)*, pages 88–95, April 2007.

[144] Afonso Ferreira, Alfredo Goldman et Julian Monteiro : Performance evaluation of routing protocols for manets with known connectivity patterns using evolving graphs. *Wireless Networks*, 16(3):627–640, 2010.

[145] Amin Vahdat, David Becker *et al.* : Epidemic routing for partially connected ad hoc networks. Rapport technique, Duke University, 2000.

[146] L.E. Parker : Alliance : an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[147] Tracy Camp, Jeff Boleng et Vanessa Davies : A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, 2(5):483–502, 2002.

[148] F. Bai, Narayanan Sadagopan et A. Helmy : Important : a framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks. *In Twenty-Second Annual Joint Conference of the IEEE Computer and Communications (INFOCOM 2003)*, volume 2, pages 825–835, March 2003.

[149] Junfei Xie, Yan Wan, J.H. Kim, Shengli Fu et K. Namuduri : Analysis of mobility models for airborne networks. *In IEEE Military Communications Conference 2013 (MILCOM 2013)*, pages 858–863, November 2013.

[150] Junfei Xie, Yan Wan, J.H. Kim, Shengli Fu et K. Namuduri : A survey and analysis of mobility models for airborne networks. *IEEE Communications Surveys Tutorials*, 16(3):1221–1238, 2014.

[151] K. Viswanath et K. Obraczka : An adaptive approach to group communications in multi hop ad hoc networks. *In Proceedings of the Seventh International Symposium on Computers and Communications (ISCC 2002)*, pages 559–566, 2002.

[152] K. Obraczka, G. Tsudik et K. Viswanath : Pushing the limits of multicast in ad hoc networks. *In 21st International Conference on Distributed Computing Systems*, pages 719–722, April 2001.

[153] Wei Lou et Jie Wu : Toward broadcast reliability in mobile ad hoc networks with double coverage. *IEEE Transactions on Mobile Computing*, 6(2):148–163, February 2007.

[154] Amit Jardosh, Elizabeth M. Belding-Royer, Kevin C. Almeroth et Subhash Suri : Towards realistic mobility models for mobile ad hoc networks. *In Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom '03)*, pages 217–229. ACM, 2003.

[155] A.P. Jardosh, E.M. Belding-Royer, K.C. Almeroth et S. Suri : Real-world environment models for mobile network evaluation. *Selected Areas in Communications, IEEE Journal on*, 23(3):622–632, March 2005.

[156] G.-C.A. Nze et F. Guinand : Mean degree of ad hoc networks in environments with obstacles. *In IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2012)*, pages 381–388, October 2012.

[157] Christian Bettstetter, Hannes Hartenstein et Xavier Pérez-Costa : Stochastic properties of the random waypoint mobility model. *Wireless Networks*, 10(5):555–567, September 2004.

[158] C. Bettstetter, G. Resta et P. Santi : The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(3):257–269, July 2003.

[159] N. Aschenbruck, E. Gerhards-Padilla et P. Martini : A survey on mobility models for performance analysis in tactical mobile networks. *Journal of Telecommunications and Information Technology*, 2:54–61, 2008.

[160] S.M. Mousavi, H.R. Rabiee, M. Moshref et A. Dabirmoghaddam : Mobisim : A framework for simulation of mobility models in mobile ad-hoc networks. *In Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMOB 2007)*, October 2007.

[161] Giovanni Resta et Paolo Santi : An analysis of the node spatial distribution of the random waypoint mobility model for ad hoc networks. *In Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC '02)*, pages 44–50. ACM, 2002.

[162] Karl Pearson : The problem of the random walk. *Nature*, 72(1865):294, 1905.

[163] Christian Bettstetter : Mobility modeling in wireless networks : Categorization, smooth movement, and border effects. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(3):55–66, July 2001.

[164] P. Nain, D. Towsley, Benyuan Liu et Zhen Liu : Properties of random direction models. *In 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, volume 3, pages 1897–1907, March 2005.

[165] Yan Wan, Kamesh Namuduri, Yi Zhou, Dayin He et Shengli Fu : A smooth-turn mobility model for airborne networks. *In Proceedings of the first ACM MobiHoc workshop on Airborne Networks and Communications (Airborne '12)*, pages 25–30, 2012.

[166] Wei Wang, Xiaohong Guan, Beizhan Wang et Yaping Wang : A novel mobility model based on semi-random circular movement in mobile ad hoc networks. *Information Sciences : an International Journal*, 180(3):399–413, February 2010.

[167] Yan Wan, K. Namuduri, Yi Zhou et Shengli Fu : A smooth-turn mobility model for airborne networks. *IEEE Transactions on Vehicular Technology*, 62(7):3359–3370, September 2013.

[168] Junfei Xie, Yan Wan, K. Namuduri, Shengli Fu, G.L. Peterson et J.F. Raquet : Estimation and validation of the 3d smooth-turn mobility model for airborne networks. *In IEEE Military Communications Conference (MILCOM 2013)*, pages 556–561, November 2013.

[169] Ouns Bouachir, Alinoe Abrassart, Fabien Garcia, Nicolas Larrieu *et al.* : A mobility model for uav ad hoc network. *ICUAS'14 Proceedings*, 2014.

[170] K. Sampigethaya, R. Poovendran, S. Shetty, T. Davis et Chuck Royalty : Future e-enabled aircraft communications and security : The next 20 years and beyond. *Proceedings of the IEEE*, 99(11):2040–2055, November 2011.

[171] S. Shirazipourazad, P. Ghosh et A. Sen : On connectivity of airborne networks in presence of region-based faults. *In IEEE Military Communications Conference 2011 (MILCOM 2011)*, pages 1997–2002, November 2011.

[172] Shahrzad Shirazipourazad, Pavel Ghosh et Arunabha Sen : On connectivity of airborne networks with unpredictable flight path of aircrafts. *In Proceedings of the First ACM MobiHoc Workshop on Airborne Networks and Communications (Airborne '12)*, pages 1–6, 2012.

[173] C.N. Ververidis et G.C. Polyzos : Service discovery for mobile ad hoc networks : a survey of issues and techniques. *IEEE Communications Surveys Tutorials*, 10(3):30–45, March 2008.

[174] A.N. Mian, R. Baldoni et R. Beraldi : A survey of service discovery protocols in multihop mobile ad hoc networks. *IEEE Pervasive Computing*, 8(1):66–74, January 2009.

[175] Jian Su et Wei Guo : A survey of service discovery protocols for mobile ad hoc networks. *In International Conference on Communications, Circuits and Systems 2008 (ICCCAS 2008)*, pages 398–404, May 2008.

[176] F.M. Anwar, Seung wha Yoo et Ki-Hyung Kim : Survey on service discovery for wireless sensor networks. *In Second International Conference on Ubiquitous and Future Networks (ICUFN 2010)*, pages 17–21, 2010.

[177] OASIS Standard : *UDDI*, 2000. http://uddi.xml.org.

[178] Sun et Apache : *JINI/River*, 1998. http://river.apache.org.

[179] Vaskar Raychoudhury, Jiannong Cao, Weigang Wu, Yi Lai, Canfeng Chen et Jian Ma : K-directory community : Reliable service discovery in manet. *In* Krishna Kant, SriramV. Pemmaraju, KrishnaM. Sivalingam et Jie Wu, éditeurs : *Distributed Computing and Networking*, volume 5935 de *Lecture Notes in Computer Science*, pages 420–433. Springer, 2010.

[180] Ulaş C Kozat et Leandros Tassiulas : Service discovery in mobile ad hoc networks : an overall perspective on architectural choices and network layer support issues. *Ad Hoc Networks*, 2(1):23–44, 2004.

[181] Francoise Sailhan et Valerie Issarny : Scalable service discovery for manet. *In Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications (PERCOM '05)*, pages 235–244, 2005.

[182] Michael Klein et Birgitta König-Ries : Multi-layer clusters in ad-hoc networks - an approach to service discovery. *In* Enrico Gregori, Ludmila Cherkasova, Gianpaolo Cugola, Fabio Panzieri et GianPietro Picco, éditeurs : *Web Engineering and Peer-to-Peer Computing*, volume 2376 de *Lecture Notes in Computer Science*, pages 187–201. Springer, 2002.

[183] M. Klein, B. Konig-Ries et P. Obreiter : Service rings - a semantic overlay for service discovery in ad hoc networks. *In Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pages 180–185, September 2003.

[184] Gregor Schiele, Christian Becker et Kurt Rothermel : Energy-efficient cluster-based service discovery for ubiquitous computing. *In Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, EW 11, 2004.

[185] Elena Meshkova, Janne Riihijärvi, Marina Petrova et Petri Mähönen : A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks*, 52(11):2097–2128, 2008.

[186] Zhiyu Liu, Ruifeng Yuan, Zhenhua Li, Hongxing Li et Guihai Chen : Survive under high churn in structured p2p systems : Evaluation and strategy. *In* VassilN. Alexandrov, GeertDick van Albada, PeterM.A. Sloot et Jack Dongarra, éditeurs : *Computational Science - ICCS 2006*, volume 3994 de *Lecture Notes in Computer Science*, pages 404–411. Springer, 2006.

[187] Eng Keong Lua, J. Crowcroft, M. Pias, R. Sharma et S. Lim : A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys Tutorials*, 7(2):72–93, Second Quarter 2005.

[188] P. Maymounkov et D. Mazieres : Kademlia : A peer-to-peer information system based on the xor metric. *Peer-to-Peer Systems*, pages 53–65, 2002.

[189] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek et H. Balakrishnan : Chord : A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[190] T. Tiendrebeogo, D. Ahmat et D. Magoni : Reliable and scalable distributed hash tables harnessing hyperbolic coordinates. *In 5th International Conference on New Technologies, Mobility and Security (NTMS 2012)*, pages 1–6, May 2012.

[191] Liang Cheng : Service advertisement and discovery in mobile ad hoc networks. *In Proceedings of CSCW*, 2002.

[192] Uday Mohan, KevinC. Almeroth et ElizabethM. Belding-Royer : Scalable service discovery in mobile ad hoc networks. *In NETWORKING 2004. Networking Technologies, Services, and Protocols ; Performance of Computer and Communication Networks ; Mobile and Wireless Communications*, volume 3042 de *Lecture Notes in Computer Science*, pages 137–149. Springer Berlin Heidelberg, 2004.

[193] M. Nidd : Service discovery in deapspace. *Personal Communications, IEEE*, 8(4):39–45, August 2001.

[194] J.L. Jodra, M. Vara, J.M. Cabero et J. Bagazgoitia : Service discovery mechanism over olsr for mobile ad-hoc networks. *In 20th International Conference on Advanced Information Networking and Applications (AINA 2006)*, volume 2, April 2006.

[195] Jeroen Hoebeke, Ingrid Moerman, Bart Dhoedt et Piet Demeester : Analysis of decentralized resource and service discovery mechanisms in wireless multi-hop networks. *In Wired/Wireless Internet Communications*, volume 3510 de *Lecture Notes in Computer Science*, pages 181–191. Springer Berlin Heidelberg, 2005.

[196] GertjanP. Halkes, Aline Baggio et KoenG. Langendoen : A simulation study of integrated service discovery. *In* Paul Havinga, Maria Lijding, Nirvana Meratnia et Maarten Wegdam, éditeurs : *Smart Sensing and Context*, volume 4272 de *Lecture Notes in Computer Science*, pages 39–53. Springer, 2006.

[197] C.N. Ververidis et G.C. Polyzos : Routing layer support for service discovery in mobile ad hoc networks. *In Third IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops)*, pages 258–262, March 2005.

[198] C.N. Ververidis et G.C. Polyzos : Extended zrp : a routing layer based service discovery protocol for mobile ad hoc networks. *In Second Annual International Conference on Mobile and Ubiquitous Systems : Networking and Services (MobiQuitous 2005)*, pages 65–72, July 2005.

[199] S. Helal, N. Desai, V. Verma et Choonhwa Lee : Konark - a service discovery and delivery protocol for ad-hoc networks. *In IEEE Wireless Communications and Networking 2003 (WCNC 2003)*, volume 3, pages 2107–2113, March 2003.

[200] Choonhwa Lee, A. Helal, N. Desai, V. Verma et B. Arslan : Konark : A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services. *IEEE Transactions on Systems, Man and Cybernetics, Part A : Systems and Humans*, 33(6):682–696, November 2003.

[201] Olga Ratsimor, Dipanjan Chakraborty, Anupam Joshi et Timothy Finin : Allia : Alliance-based service discovery for ad-hoc environments. *In Proceedings of the 2nd international workshop on Mobile commerce*, pages 1–9. ACM, 2002.

[202] Li Li et L. Lamont : A lightweight service discovery mechanism for mobile ad hoc pervasive environment using cross-layer design. *In Third IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops)*, pages 55–59, March 2005.

[203] Paul Mockapetris : *Domain names - concepts and facilities*, November 1987. RFC 1034.

[204] Paul Mockapetris : *Domain names - implementation and specification*, November 1987. RFC 1035.

[205] Ioannis Partsakoulakis et George Vouros : Roles in mas. *In* ThomasA. Wagner, éditeur : *An Application Science for Multi-Agent Systems*, volume 10 de *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 133–154. Springer, 2004.

[206] G. Ayorkor Korsah, Anthony Stentz et M. Bernardine Dias : A comprehensive taxonomy for multi-robot task allocation. *International Journal of Robotics Research*, 32(12):1495–1512, October 2013.

[207] Brian P Gerkey et Maja J Matarić : A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[208] Brian P Gerkey et Maja J Mataric : Multi-robot task allocation : Analyzing the complexity and optimality of key architectures. *In IEEE International Conference on Robotics and Automation (ICRA'03)*, volume 3, pages 3862–3868, 2003.

[209] G. Antonelli, F. Arrichiello et S. Chiaverini : The null-space-based behavioral control for mobile robots. *In IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2005)*, pages 15–20, June 2005.

[210] Javier de Lope, Darío Maravall et Yadira Quiñonez : Response threshold models and stochastic learning automata for self-coordination of heterogeneous multi-task distribution in multi-robot systems. *Robotics and Autonomous Systems*, 61(7):714–720, 2013.

[211] M.B. Dias, Robert Zlot, N. Kalra et A. Stentz : Market-based multirobot coordination : A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, July 2006.

[212] Xiao Jia et M.Q.-H. Meng : A survey and analysis of task allocation algorithms in multi-robot systems. *In IEEE International Conference on Robotics and Biomimetics (ROBIO 2013)*, pages 2280–2285, December 2013.

[213] Lingzhi Luo, N. Chakraborty et K. Sycara : Distributed algorithm design for multi-robot task assignment with deadlines for tasks. *In IEEE International Conference on Robotics and Automation 2013 (ICRA 2013)*, pages 3007–3013, May 2013.

[214] T.S. Dahl, M.J. Mataric et G. Sukhatme : Multi-robot task-allocation through vacancy chains. *In IEEE International Conference on Robotics and Automation 2003 (ICRA '03)*, volume 2, pages 2293–2298, September 2003.

[215] Javier de Lope, Darío Maravall et Yadira Quiñonez : Response threshold models and stochastic learning automata for self-coordination of heterogeneous multi-task

distribution in multi-robot systems. *Robotics and Autonomous Systems*, 61(7):714–720, 2013.

[216] Robert Michael Zlot : *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams*. Thèse de doctorat, Robotics Institute, Carnegie Mellon University, December 2006.

[217] GeorgeC. Polyzos, ChristopherN. Ververidis et EliasC. Efstathiou : Service discovery and provision for autonomic mobile computing. *In Autonomic Communication*, volume 3854 de *Lecture Notes in Computer Science*, pages 226–236. 2006.

[218] D Mills : Modelling and analysis of computer network clocks. *Electrical Engineering Department Report*, 9252, 1992.

[219] Baljinder Kaur et Amandeep Kaur : A survey of time synchronization protocols for wireless sensor networks. *In International Journal of Computer Science and Mobile Computing*, volume 2, pages 100–106. IJCSMC, 2013.

[220] D. Mills, U. Delaware, J. Martin, J. Burbank et W. Kasch : *Network Time Protocol Version 4 : Protocol and Algorithms Specification*, June 2010. RFC 5905.

[221] D.C. Jefferson, S.M. Lichten et L.E. Young : A test of precision gps clock synchronization. *In Proceedings of the 50th IEEE International Frequency Control Symposium*, pages 1206–1210, June 1996.

[222] S.B. Moon, P. Skelly et D. Towsley : Estimation and removal of clock skew from network delay measurements. *In INFOCOM Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 227–234, March 1999.

[223] Bruno Bassano Irène Girard, Christopher Adrados et Georges Janeau : Application de la technologie gps au suivi du déplacement de bouquetins des alpes. (XXIV):105–126, 2009.

[224] Bharath Sundararaman, Ugo Buy et Ajay D Kshemkalyani : Clock synchronization for wireless sensor networks : a survey. *Ad Hoc Networks*, 3(3):281–323, 2005.

[225] I. Litovsky, Y. Métivier et E. Sopena : Different local controls for graph relabeling systems. *Mathematical systems theory*, 28(1):41–65, 1995.

[226] Jérémie Chalopin et Yves Métivier : A Bridge Between the Asynchronous Message Passing Model and Local Computations in Graphs. *In International Symposium on Mathematical Foundations of Computer Science (MFCS 2005)*, volume 3618 de *Lecture notes in computer science*, pages 212–223. Springer, August 2005.

[227] Igor Litovsky, Yves Métivier et Éric Sopena : Handbook of graph grammars and computing by graph transformation. chapitre Graph Relabelling Systems and Distributed Algorithms, pages 1–56. World Scientific Publishing Co., Inc., 1999.

[228] Jérémie Chalopin : *Algorithmique Distribuée, Calculs Locaux et Homomorphismes de Graphes*. Thèse de doctorat, Université Sciences et Technologies - Bordeaux I, November 2006.

[229] Arnaud Casteigts et Serge Chaumette : Dynamicity aware graph relabeling systems (da-grs), a local computation based model to describe manet algorithms. *In Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS)*, pages 231–236, 2005.

[230] Arnaud Casteigts : *Contribution à l'algorithmique distribuée dans les réseaux mobiles ad hoc*. Thèse de doctorat, Université Sciences et Technologies - Bordeaux I, September 2007.

[231] Rémi Laplace : *Applications et services DTN pour flotte collaborative de drones*. Thèse de doctorat, Université Sciences et Technologies - Bordeaux I, December 2012.

[232] T.R. Henderson, M. Lacage, G.F. Riley, C. Dowell et JB Kopena : Network simulations with the ns-3 simulator. *ACM SIGCOMM demonstration*, 2008.

[233] A. Varga *et al.* : The omnet++ discrete event simulation system. *In Proceedings of ESM*, volume 9, 2001.

[234] Arnaud Casteigts : The jbotsim library. *CoRR*, abs/1001.1435, 2010.

[235] Arnaud Casteigts : *JBotSim*. http://jbotsim.sourceforge.net.

[236] B. Yamini et D.V. Selvi : Cloud virtualization : A potential way to reduce global warming. *In Proceedings of IEEE RSTSCC*, pages 55–57, nov. 2010.

[237] F. Bellard : QEMU, a fast and portable dynamic translator. *In Proceedings of USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[238] N.M.M.K. Chowdhury et R. Boutaba : Network virtualization : state of the art and research challenges. *IEEE Communications Magazine*, 47(7):20–26, 2009.

[239] P. Mell et T. Grance : The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 2009.

[240] KVM : *Virtio*. http://www.linux-kvm.org/page/Virtio.

[241] NBD : *Network Block Device*. http://nbd.sourceforge.net.

[242] NetFilter : *NetFilter*, 2000. http://www.netfilter.org.

[243] B. Hubert, G. Maxwell, R. Van Mook, M. Van Oosterhout, P.B. Schroeder et J. Spaans : Linux advanced routing & traffic control. *In Ottawa Linux Symposium*, pages 213–222, 2003.

[244] Robert Shingledecker : *TinyCore Linux*, 2008. http://tinycorelinux.net.

[245] Renzo Davoli : Vde : Virtual distributed ethernet. *In Proc. of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pages 213–220, 2005. http://vde.sourceforge.net.

[246] Damien Magoni : *Virtual Network Device*, 2013. http://www.labri.fr/perso/magoni/vnd.

[247] TcpDump : *LibPcap*, 1987. http://www.tcpdump.org.

[248] Graphviz : *Graph Visualization Software*, 1988. http://www.graphviz.org.

[249] Damien Magoni : *Network Mobilizer*, 2012. http://www.labri.fr/perso/magoni/nemo.

[250] Nils Aschenbruck, Raphael Ernst, Elmar Gerhards-Padilla et Matthias Schwamborn : Bonnmotion : A mobility scenario generation and analysis tool. *In Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools '10)*, pages 1–10. ACM, 2010.

[251] Bob Lantz, Brandon Heller et Nick McKeown : A network in a laptop : Rapid prototyping for software-defined networks. *In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*, pages 19 :1–19 :6, New York, NY, USA, 2010. ACM.

[252] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz et Nick McKeown : Reproducible network experiments using container-based emulation. *In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, pages 253–264, New York, NY, USA, 2012. ACM.

[253] Daniel Lezcano : *LXC*, 2008. http://lxc.sourceforge.net.

[254] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado et S. Shenker : Extending networking into the virtualization layer. *Proceedings of ACM HotNets workshop*, october 2009.

[255] Keith Winstein et Hari Balakrishnan : Mosh : An interactive remote shell for mobile clients. *In Presented as part of the 2012 USENIX Annual Technical Conference*, pages 177–182, Berkeley, CA, 2012. USENIX.

[256] Stanford University : *Evaluation of Mosh performance results.* http://reproducingnetworkresearch.wordpress.com/2013/03/13/cs244-2013-evaluation-of-mosh-mobile-shell-performance-results.

[257] Yongguo Mei, Yung-Hsiang Lu, Y.C. Hu et C.S.G. Lee : A case study of mobile robot's energy consumption and conservation techniques. *In ICAR 12th*, pages 492–497, July 2005.

[258] T. Imielinski et J. Navas : *GPS-Based Addressing and Routing.* IETF, November 1996. rfc 2009.

[259] J. Polk, J. Schnizlein et M. Linsner : *Dynamic Host Configuration Protocol Option for Coordinate-based Location Configuration Information.* IETF, July 2008. rfc 3825.

[260] Open Networking Foundation : Software-defined networking : The new norm for networks. Rapport technique, April 2012. White paper.